DOCUMENTATIE

TEMA *2*

NUME STUDENT: Filip Iarina Eden
GRUPA: 30221

CUPRINS

1. Obiectivul temei

(i) Obiectivul principal

Obiectivul principal al temei este de a proiecta și implementa o aplicație de gestionare a cozilor care atribuie clienților cozi astfel încât timpul de așteptare să fie minimizat.

(ii) Obiectivul secundar

1.Proiectarea unui model de simulare a cozilor și a clienților (Capitolul 1: Introducere)
2.Generarea unui set de N clienți caracterizați prin ID, timpul de sosire și durata de servire (Capitolul 2: Generarea de clienți)
3.Implementarea unei strategii de alocare a clienților la cozi cu timp minim de așteptare (Capitolul 3: Strategii de alocare a coziilor)
4.Dezvoltarea unui algoritm de simulare a comportamentului clienților în cozi și de calcul al timpului total petrecut de fiecare client în cozi (Capitolul 4: Algoritm de simulare a cozilor și de calcul al timpului total petrecut în cozi)
5.Calculul timpului mediu de așteptare pentru toți clienții (Capitolul 5: Calculul timpului mediu de așteptare)

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

1.Analiza problemei:

Problema necesită proiectarea și implementarea unei aplicații de gestionare a cozilor care atribuie clienții la cozi astfel încât timpul de așteptare al acestora să fie minimizat. Aceasta este o problemă obișnuită care apare în multe situații din lumea reală, cum ar fi într-o bancă, un spital sau un aeroport, în care oamenii trebuie să aștepte la coadă să fie rândul lor pentru a primi servicii.

Pentru a minimiza timpul de așteptare al clienților, o soluție este adăugarea mai multor servere sau cozi în sistem, dar această abordare crește costul furnizorului de servicii. Prin urmare, aplicația de gestionare a cozilor trebuie să simuleze o serie de N clienți care sosesc pentru service, care intră în Q cozi, așteaptă, sunt servite și, în final, părăsesc cozile, menținând în același timp numărul de cozi la minim.

Aplicația trebuie să urmărească timpul total petrecut de fiecare client în cozi și să calculeze timpul mediu de așteptare. Clienții sunt generați la începutul simulării și sunt caracterizați de trei parametri: ID, sosire (timpul de simulare când sunt gata să intre în coadă) și tservice (interval de timp sau durata necesară pentru deservirea clientului, adică timpul de așteptare). când clientul se află în fața cozii).

Aplicația ar trebui să adauge fiecare client la coadă cu timpul minim de așteptare atunci când timpul de sosire a acestuia este mai mare sau egal cu timpul de simulare. Aceasta înseamnă că clienții ar trebui să fie serviți într-o manieră primul venit, primul servit, iar timpul de așteptare ar trebui redus la minimum prin alocarea clienților la cozile cu cel mai scurt timp de așteptare.

2.Modele și scenarii:

Pentru a modela aplicația de gestionare a cozilor, putem folosi următoarele entități:
Task: reprezintă un client care trebuie deservit
Server: reprezintă un server care servește clienții dintr-o coadă

Putem folosi următoarele scenarii pentru a descrie comportamentul aplicației:
Inițializare: generați N clienți cu sosire aleatoare și valori tservice.
Simulare: pentru fiecare pas de timp de simulare, faceți următoarele:
Verificați dacă vreun client a sosit și adăugați-l la coadă cu timpul minim de așteptare.
Serviți clienții din cozi prin scăderea valorii lor tservice.
Eliminați clienții din cozile care au fost servite complet.
Încetarea: Calculați timpul mediu de așteptare al clienților.

3.Cazuri de utilizare

Adăugarea unui client la o coadă: in acest caz de utilizare se va adauga clientul la serverul cu timpul minim de așteptare atunci când timpul de sosire al acestuia este mai mare sau egal cu timpul de simulare.

Eliminarea unui client dintr-o coadă: acest caz de utilizare permite utilizatorului să elimine un client dintr-o coadă, fie pentru că clientul a fost servit, fie a părăsit coada. Sistemul va urmări timpul total petrecut de fiecare client în cozi și va calcula timpul mediu de așteptare.

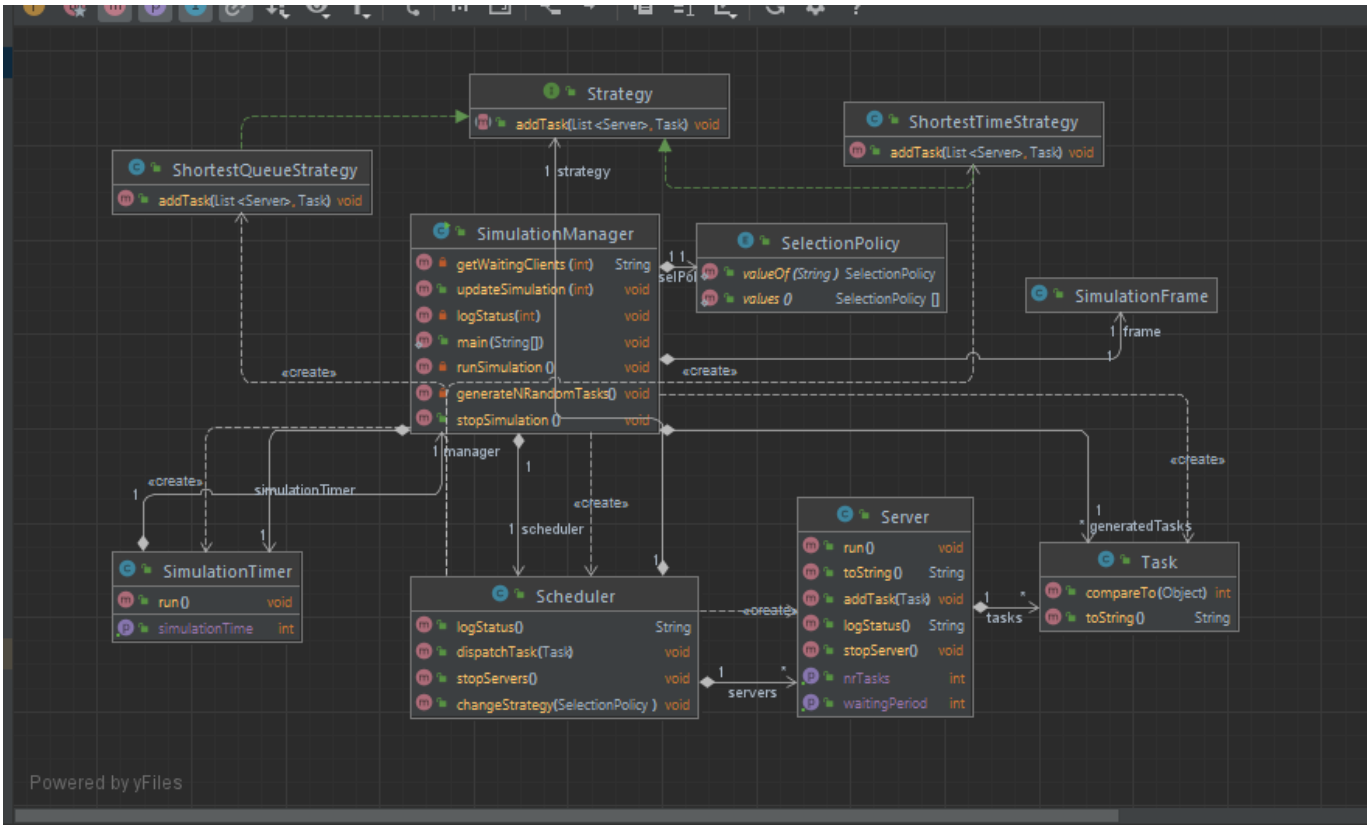
Vizualizați starea cozii: acest caz de utilizare permite utilizatorului să vadă starea curentă a unei cozi, inclusiv numărul de clienți care așteaptă în coadă, ID-urile acestora și timpul estimat de așteptare pentru fiecare client.

Adăugați o nouă coadă: Acest caz de utilizare permite utilizatorului să adauge o nouă coadă la sistem, ceea ce va crește capacitatea aplicației de gestionare a cozii și va reduce timpul de așteptare pentru clienți.

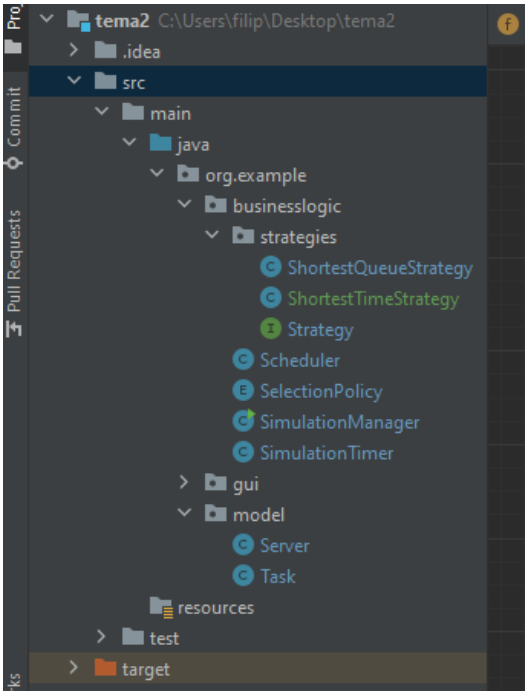
Eliminați o coadă: Acest caz de utilizare permite utilizatorului să elimine o coadă din sistem, ceea ce va reduce capacitatea aplicației de gestionare a cozii și va crește timpul de așteptare pentru clienți.

Simulați timpii de sosire și de service ale clienților: Acest caz de utilizare permite utilizatorului să simuleze orele de sosire și de service ale clienților pentru o anumită perioadă, pentru a testa eficiența aplicației de gestionare a cozilor și a optimiza numărul de cozi necesare pentru a minimiza timpul de așteptare.

3. Proiectare



Pachetele cu clasele :



4. Implementare

I. In pachetul businesslogic:

1.Clasa Scheduler:

Această clasă reprezintă planificatorul pentru o aplicație de management al cozilor, ce atribuie sarcini la servere în așa fel încât timpul de așteptare să fie minimizat. Planificatorul are rolul de a adăuga sarcinile în coada serverului cu timpul minim de așteptare.

Atribuțiile clasei Scheduler includ:
Inițializarea și crearea serverelor în momentul creării instanței de Scheduler.
Dispatch-ul sarcinilor la servere folosind o strategie de planificare specifică (ShortestQueueStrategy).
Oferirea posibilității de oprire a serverelor.
Generarea unui raport cu statusul tuturor serverelor, prin apelarea metodei logStatus().
Mai precis, metoda dispatchTask() primește ca parametru o sarcină (Task) și o adaugă la coada serverului care are timpul minim de așteptare conform strategiei de planificare.
Metoda stopServers() oprește toate serverele din lista de servere a planificatorului.
Metoda logStatus() returnează un șir de caractere care reprezintă statusul tuturor serverelor din lista, prin concatenarea rapoartelor de status generate de metoda logStatus() a fiecărui server.

2.Clasa SimulatorManager

Această clasă reprezintă managerul de simulare al aplicației. Aici se fac setările pentru parametrii simulării și se rulează simularea în sine.

Membrii clasei și funcțiile:

- timeLimit, maxProcessingTime, minProcessingTime, numberOfServers, numberOfClients, selPol: variabilele de configurare ale simulării.
- scheduler: instanța clasei Scheduler care gestionează distribuirea sarcinilor (tasks) pe servere (servers).
- frame: instanța clasei SimulationFrame care conține interfața grafică a aplicației.
- generatedTasks: lista de sarcini generate aleatoriu pentru simulare.
- SimulationTimer: instanța clasei SimulationTimer care se ocupă cu măsurarea timpului de simulare.
- runSimulation(): pornește simularea prin programarea repetată a rulării funcției updateSimulation() la fiecare 1 secundă.
- updateSimulation(): se ocupă cu actualizarea stării simulării la fiecare iterație prin adăugarea sarcinilor (tasks) în cozi și afișarea stării.
- logStatus(): afișează starea curentă a simulării la un moment dat.
- getWaitingClients(): afișează o listă de sarcini (tasks) care așteaptă să fie procesate.
- generateNRandomTasks(): generează aleatoriu sarcini (tasks) cu parametrii specificați și le sortează după timpul de sosire (tArrival).
- main(): pornește aplicația prin crearea unei instanțe a clasei SimulationManager.
- stopSimulation(): încetează simularea, dar nu face nimic în această implementare.

În esență, această clasă coordonează simularea și interacțiunea între diversele componente ale aplicației (generatorul de sarcini, planificatorul de sarcini și serverele care procesează sarcinile).

3.Clasa SimulationTime:

Această clasă implementează un cronometru de simulare, care se execută la intervale regulate de timp. Are o referință la clasa SimulationManager pentru a putea apela metoda updateSimulation(), care actualizează starea simulării în funcție de timpul curent al simulării.

Când este creată o instanță a clasei, se setează limitele de timp pentru simulare și se primește o referință la un Timer, care este folosit pentru a planifica și efectua actualizările cronometrate. Metoda run() se execută la fiecare interval de timp specificat și actualizează starea simulării prin apelul manager.updateSimulation(). Dacă timpul de simulare a atins limita maximă, timer-ul este oprit și simularea se termină.

4.Enum SelectionPolicy

Acest enum numit SelectionPolicy definește politica de selecție pentru gestionarea sarcinilor în cadrul sistemului simulat. În acest moment, singura politică definită este SHORTEST_TIME, care alege serverul cu cel mai mic timp de procesare rămas pentru a prelua o sarcină.

I. In pachetul strategies:

1.Clasa ShortestQueueStrategy:

Clasa ShortestQueueStrategy implementează interfața Strategy și reprezintă o strategie de adăugare a sarcinilor în cadrul unui sistem de cozi cu servere. Mai exact, metoda addTask are rolul de a găsi serverul cu cel mai mic număr de sarcini în coadă și de a adăuga sarcina dată ca parametru în coada acelui server.

Pentru a găsi serverul cu cel mai mic număr de sarcini în coadă, clasa parcurge lista de servere dată ca parametru și compara numărul de sarcini din coada fiecărui server cu un număr minim (min) inițializat cu o valoare mare. Dacă numărul de sarcini din coada serverului curent este mai mic decât valoarea minimă curentă, atunci se actualizează valoarea minimă și se reține referința către acel server ca fiind serverul țintă. După aceea, sarcina dată ca parametru este adăugată în coada serverului țintă, iar un mesaj de confirmare este afișat în consolă.

2.Clasa ShortestTimeStrategy:

Această clasă implementează interfața Strategy și definește metoda addTask care adaugă o nouă sarcină la serverul cu cel mai mic waitingPeriod.

În primul rând, se initializează o variabilă minWaitingPeriod cu o valoare mare (pentru a fi siguri că va fi depășită de valoarea waitingPeriod a primului server). De asemenea, se inițializează o variabilă targetServer cu valoarea null, care va fi utilizată pentru a ține serverul cu cel mai mic waitingPeriod.

Apoi, se iterază prin lista de servere și pentru fiecare server se verifică dacă waitingPeriod-ul său este mai mic decât valoarea curentă a variabilei minWaitingPeriod. Dacă da, atunci serverul respectiv devine ținta și se actualizează minWaitingPeriod și targetServer.

La final, se adaugă sarcina la targetServer și se afișează un mesaj care confirmă adăugarea sarcinii la serverul ales.

3.Interfata Strategy

Aceasta interfață numită Strategy este folosită pentru a implementa diferite strategii de adăugare a sarcinilor (Task) la servere (Server). Interfața definește o singură metodă addTask(List<Server> servers, Task task), care primește o listă de servere și o sarcină și adaugă sarcina într-un anumit server folosind o anumită strategie specificată de implementarea concretă a interfeței. Această interfață este folosită în cadrul clasei Scheduler pentru a alege ce strategie să fie utilizată în funcție de politica de selectare specificată.

II. In pachetul model

1.Clasa Server:

Clasa Server reprezintă o entitate server în simulare. Implementează interfața Runnable, ceea ce înseamnă că poate fi executată simultan într-un fir separat. Serverul are un BlockingQueue de obiecte Task, care reprezintă sarcinile care așteaptă să fie procesate. Metoda addTask a serverului adaugă o nouă sarcină la coadă, iar metoda getNrTasks returnează numărul de sarcini aflate în prezent în coadă.

Metoda de rulare este logica principală a Serverului. Se rulează într-o buclă până când indicatorul isStopAsked este setat la adevărat. Bucla preia o sarcină din coadă utilizând metoda take (care blochează dacă coada este goală), o procesează prin somn pentru perioada de timp tService (simulând timpul necesar procesării sarcinii) și decrește contorul waitingPeriod prin câte unul pentru fiecare secundă pe care serverul așteaptă. În cele din urmă, imprimă un mesaj pe consolă că sarcina a fost procesată.

Metoda toString returnează o reprezentare șir a serverului, inclusiv BlockingQueue de sarcini și waitingPeriod. Metoda stopServer setează indicatorul isStopAsked la true, ceea ce semnalează metodei de rulare să iasă din buclă și să oprească serverul. În cele din urmă, metoda logStatus returnează o reprezentare în șir a stării serverului, fie „închis” dacă nu există sarcini în coadă, fie o reprezentare în șir a fiecărei sarcini din coadă.

2. Clasa Task:

Clasa Task este o clasa simpla care reprezinta o sarcina ce trebuie indeplinita in cadrul unei simulari de tip coada de asteptare la un server.

Aceasta contine trei atribute:

ID - reprezinta identificatorul unic al sarcinii
tArrival - timpul la care sarcina a ajuns la server
tService - timpul necesar pentru a finaliza sarcina

Clasa implementeaza interfata Comparable pentru a permite compararea sarcinilor in functie de timpul la care au ajuns la server. De asemenea, clasa suprascrie metoda toString pentru a afisa informatii despre o sarcina intr-un format usor de citit.

5. Concluzii

- Am înțeles importanța gestionării cozilor în sistemele bazate pe ele și obiectivul principal de a minimiza timpul de așteptare al clienților.
- Am învățat că există o serie de parametrii și condiții care trebuie luate în considerare pentru a proiecta și implementa o astfel de aplicație, cum ar fi caracteristicile clienților, timpul de simulare și condițiile de adăugare a clienților în cozi.
- Am realizat că adăugarea de mai multe cozi în sistem poate minimiza timpul de așteptare, dar poate crește și costurile furnizorului de servicii.
- Am dezvoltat o aplicație de gestionare a cozilor care îndeplinește cerințele de proiectare și care a fost capabilă să simuleze cu succes și să monitorizeze clienții care se adaugă, așteaptă, sunt serviți și pleacă din cozi.

Posibile dezvoltări ulterioare:

- Adăugarea unor funcționalități precum posibilitatea de a adăuga și elimina cozi și de a gestiona resursele disponibile.
- Optimizarea timpului de așteptare prin utilizarea unor algoritmi mai avansați și mai sofisticați pentru a atribui clienții la cozi.
- Dezvoltarea unei interfețe grafice pentru a face aplicația mai accesibilă și mai ușor de utilizat pentru utilizatorii finali.
- Implementarea de instrumente de monitorizare și analiză pentru a putea urmări și analiza performanța sistemului de cozi în timp real.

6. Bibliografie

https://dsrl.eu/courses/pt/materials/PT2023_A2_S2.pdf
https://dsrl.eu/courses/pt/materials/PT2023_A2_S1.pdf
https://dsrl.eu/courses/pt/materials/PT2023_A2.pdf
https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html
https://stackoverflow.com/questions/16252269/how-to-sort-a-list-arraylist
https://stackoverflow.com/questions/363681/how-do-i-generate-random-integers-within-a-specific-range-in-java
https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html
https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html
https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm
https://en.wikipedia.org/wiki/Strategy_pattern