DOCUMENTAȚIE

TEMA 2

NUME SI PRENUME: MOȘILĂ LUCIANA

GRUPA: 30226

CUPRINS

1.Obiectivul temei

Obiectivul principal al temei:

Proiectarea și implementarea unei aplicații de gestionare a cozilor, care distribuie clientii la cozi astfel încât timpul de așteptare să fie minimizat.

Obiective secundare:

1. Definirea claselor cu date specifice pentru stocarea informațiilor despre clienți și cozile asociate.

2. Generarea aleatoare a N clienți cu informațiile lor de identificare și caracteristicile de timp.

3. Crearea a Q cozi și metoda de procesare aferenta, folosind clasele cu date definite anterior.

4. Calcularea timpului de așteptare total pentru fiecare client și determinarea timpului mediu de așteptare.

5. Crearea unei interfețe utilizator pentru introducerea datelor de intrare.

6. Testarea aplicației cu diferite seturi de date de intrare și raportarea rezultatelor.

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

Se identifica următoarele cerințe functionale:

- Aplicația trebuie să permită simularea unui număr de N clienți care vin, intră în cozi, așteaptă, sunt serviți și părăsesc cozile.

- Fiecare client trebuie să fie caracterizat de trei parametri: ID, timpul de sosire și durata de servire.

- Aplicația trebuie să calculeze timpul total petrecut de fiecare client în cozi și să calculeze timpul mediu de așteptare.

- Fiecare client trebuie adăugat în coadă cu cel mai mic timp de așteptare atunci când timpul său de sosire este mai mare sau egal cu timpul de simulare.

Cerințele non-functionale sunt:

- Performanță: aplicația trebuie să poată procesa simularea într-un interval de timp rezonabil și să poată gestiona un număr mare de clienți și cozi.

- Ușurință de utilizare: aplicația trebuie să ofere o interfață de utilizare simplă și intuitivă pentru inserarea datelor de intrare și afișarea rezultatelor.

- Fiabilitate: aplicația trebuie să fie capabilă să gestioneze erorile și să ofere o simulare precisă și consistentă.

- Scalabilitate: aplicația trebuie să fie capabilă să gestioneze un număr variabil de cozi și clienți, fără a afecta performanța sau fiabilitatea.

Descrierea cazurilor de utilizare (use-case) pentru aplicația de gestionare a cozilor:

1. Introducerea parametrilor de intrare: Utilizatorul introduce numărul de clienți, numărul de cozi, intervalul de simulare și duratele minime și maxime de așteptare și servire a clienților.

2. Utilizarea unei interfețe prietenoase: Aplicația are o interfață intuitivă și ușor de utilizat, care permite utilizatorului să introducă parametrii de intrare și să vizualizeze informații despre clienți și cozi.

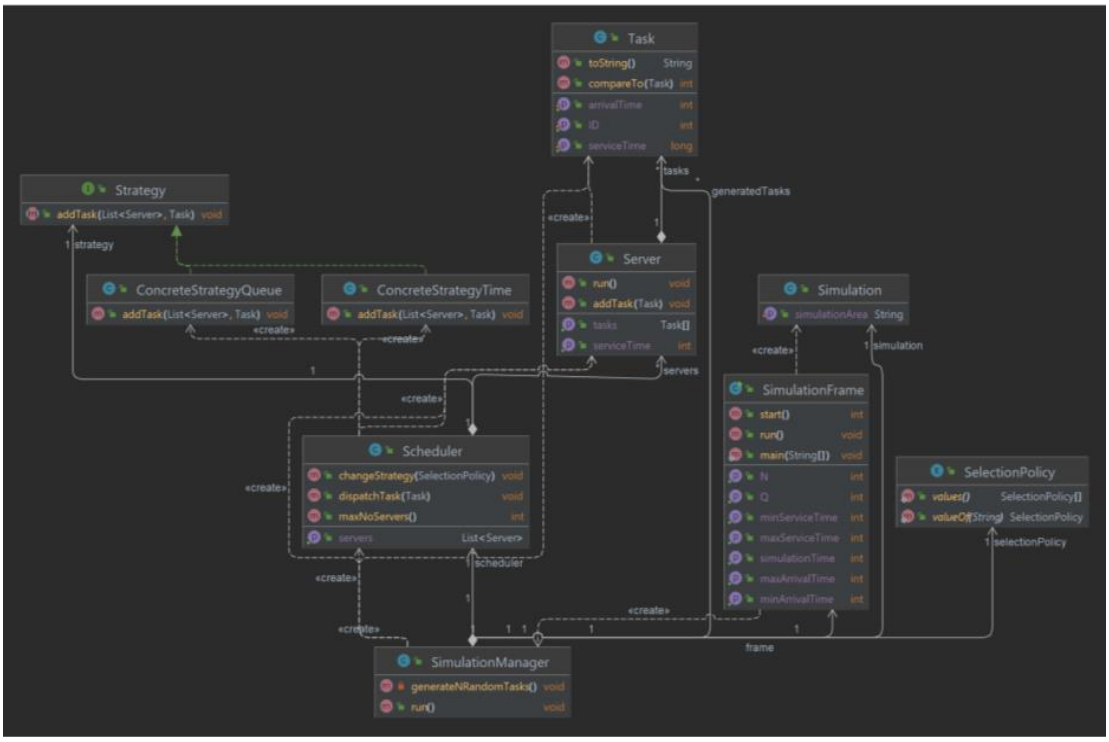
3. Generarea automată a clienților în funcție de datele de intrare: Odată ce utilizatorul a introdus parametrii de intrare, aplicația generează automat o serie de clienți cu caracteristici specifice (ID, timp de sosire și durată de servire).

4. Repartizarea unui client în funcție de cea mai scurtă coadă: Aplicația repartizează un client în coada cu cea mai mică durată de așteptare.

Scopul aplicației este de a minimiza timpul de așteptare al clienților prin optimizarea repartizării lor în cozi, astfel încât să se evite creșterea costurilor de serviciu prin adăugarea de noi cozi. Aplicația urmărește și înregistrează timpul total petrecut de fiecare client în cozi și calculează timpul mediu de așteptare.

3. Proiectare

Diagrama UML de clase este urmatoarea:



Am folosit anumite clase specifice care sunt implementate cu diferite interfete precum Runnable si Comparable.

Clasa Task este utilizata pentru a reprezenta sarcinile (tasks) ale clientilor intr-o simulare de cozi la un magazin. Datele unui client (ID, timp de sosire, timp de procesare) sunt stocate ca proprietati ale obiectului Task. Astfel, clasa Task permite incapsularea datelor clientilor si definirea unei functionalitati pentru compararea obiectelor de tip Task, necesara in cazul organizarii coziilor de asteptare in magazin.

Clasa Server implementează un server simplu care preia sarcini (Task) și le procesează în ordinea în care au fost adăugate. Principiile de programare o OOP sunt următoarele:

Encapsularea: clasa Server utilizează encapsularea pentru a ascunde detalii interne ale implementării sale. Variabilele private tasks și waitingPeriod sunt accesate numai prin metodele publice addTask, getTasks și getServiceTime, pentru a se asigura că acestea sunt modificate și accesate în mod corect.

Abstracția: clasa Server abstrage conceptul de server, care primește sarcini și le procesează. În cadrul clasei, sarcinile sunt reprezentate de obiecte de tipul Task.

4. Implementare

“Task” este o clasă de bază pentru reprezentarea sarcinilor. Are câteva câmpuri și metode importante:

-ID: identificatorul unic al sarcinii

-arrivalTime: timpul de sosire al sarcinii

-serviceTime: timpul necesar pentru a finaliza sarcina, reprezentat printr-un AtomicInteger

Metode:

-Constructorul: ia ca parametri ID-ul, timpul de sosire și timpul de serviciu și inițializează câmpurile corespunzătoare

-getter-ele și setter-ele pentru câmpurile ID, arrivalTime și serviceTime

-getServiceTime(): returnează timpul necesar pentru a finaliza sarcina ca o valoare long

-toString(): returnează o reprezentare sub forma unei șiruri de caractere a sarcinii

-compareTo(): metoda necesară pentru a implementa interfața Comparable<Task>, utilizată pentru a sorta o listă de sarcini după timpul de sosire. Returnează rezultatul comparației între timpul de sosire al acestei sarcini și timpul de sosire al altei sarcini.

Clasa "Server" are următoarele câmpuri și metode importante:

- tasks: un obiect de tip BlockingQueue<Task>, utilizat pentru a stoca sarcinile ce trebuie procesate de server

- waitingPeriod: un obiect de tip AtomicInteger, utilizat pentru a ține evidența timpului total de așteptare al tuturor sarcinilor din coadă

- addTask(Task newTask): adaugă o sarcină nouă în coada de sarcini și actualizează timpul total de așteptare

- run(): metoda care rulează în firul de execuție al serverului și preia sarcinile din coada de sarcini, procesându-le pe rând

- getTasks(): returnează toate sarcinile din coada de sarcini într-un tablou de obiecte Task

- getServiceTime(): calculează și returnează timpul total de procesare a sarcinilor din coada de sarcini.

Clasa “Scheduler” contine o lista cu obiecte de tip “Server”, un numar de tip intreg pentru numarul de cozi si o zona pentru memorarea tipului de strategie aleasa, strategie in functie de timp sau de locurile libere la coada. Este responsabila pentru crearea cozilor si pornirea unui thread pentru fiecare coada in parte. In pus avem o metoda principala, importanta pentru schimbarea tipului de strategie. Pentru strategie am implementat doua clase aditioanale: “ConcreteStrategyQueue” si “ConcreteStrategyTime” cu algoritmi care aleg ce mai buna variant de a pune clientii in functie de selectia dorita.

Alte doua clase sunt “Simulation” si “SimulationFrame”, ele sunt interfetele grafice ale proiectul realizat. In clasa SimulationFrame ne este permis sa introducem datele necesare: numarul de clienti, numarul de cozi, timpul simularii si doua intervale pentru “arrivalTime” si “serviceTime”. Clasa “Simulation” functioneaza ca o fereastra pentru afisarea datelor, in timp real.

Clasa “SimulationManager” are rolul de a porni un Thread principal care implementeaza conceptul de timp pentru simularea cozilor. In aceasta clasa este scrisa si metoda pentru generarea aleatoare a clientilor cu datele specifice lor din clasa Task. Din clasele pentru interfata grafica se culeg datele de intrare si sunt procesate si afisate in fisier dar si in clasa pentru simularea in timp real.

5.Rezultate

Am introdus urmatoarele date de intrare:

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

Rezultatele dupa rularea programului au fost scrise pentru fiecare simulare in parte intr-un fisier text cu nume specific fiecarui test: “Test1”, “Test2” si “Test3”.

6.Concluzii

A fost dezvoltată o aplicație de gestionare a cozilor care minimizează timpul de așteptare al clienților. Aceasta utilizează o simulare a timpului de execuție și un set de date de intrare specificat de utilizator pentru a distribui clienții în cozi și alocarea resurselor într-un mod optim. Strategia aplicată adaugă clienții în coada cu cel mai mic timp de așteptare atunci când timpul lor de sosire este mai mare sau egal cu timpul de simulare curent.

Din această temă, am învățat să lucrez cu thread-uri pentru a gestiona simultan clienții și cozile, să implementez o afișare live pentru a putea monitoriza desfășurarea simulării și să folosesc scrierea în fișiere în Java pentru a putea salva rezultatele obținute.

Posibile îmbunătățiri ar fi:

- Adăugarea unui sistem de prioritizare pentru clienți importanți

- Integrarea unei opțiuni de plată online

- Implementarea unui sistem de notificare a clienților pentru a reduce timpul de așteptare și pentru a oferi o experiență mai bună.

7. Bibliografie

1. https://users.utcluj.ro/~igiosan/teaching_poo.html

3. https://en.wikipedia.org/wiki/Java_(programming_language)

4. https://dsrl.eu/courses/pt/materials/PT2023_A2_S1

5. https://dsrl.eu/courses/pt/materials/PT2023_A2_S2