DOCUMENTATIE

TEMA *2*

NUME STUDENT: Ardelean Mitică-Mario
GRUPA: 30224

CUPRINS

1. Obiectivul temei

Obiectivul temei este de a proiecta și implementa o aplicație care are ca scop analiza sistemelor bazate pe cozi de așteptare prin (1) simulând o serie de N clienți care sosesc pentru servire, care intră în cozile Q, așteaptă, sunt serviți și în cele din urmă părăsesc cozile și (2) calcularea timpului mediu de așteptare, a timpului mediu de serviciu și a orelor de vârf..
Obiectivele secundare al acestei teme sunt următoarele:

- Analiza problemei și identificarea cerințelor (Capitolul 2)
- Proiectarea aplicației de simulare (Capitolul 3)
- Implementarea aplicației de simulare (Capitolul 4)
- Testarea aplicației de simulare (Capitolul 5)

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

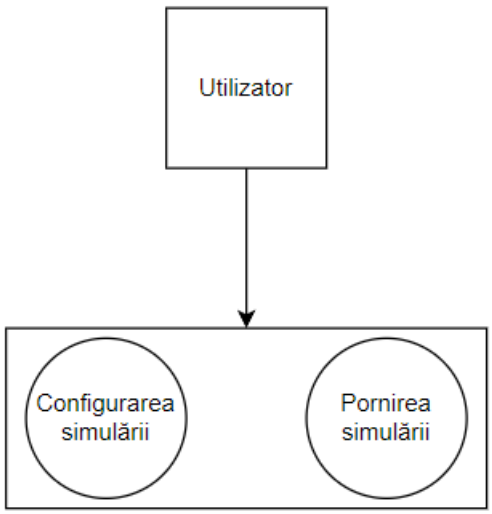
Cerințele funcționale sunt următoarele:

- Aplicația de simulare ar trebui să permită utilizatorilor să configureze simularea
- Aplicația de simulare ar trebui să permită utilizatorilor să înceapă simularea
- Aplicația de simulare ar trebui să afișeze evoluția cozilor în timp real

Cerințele non-funcționale sunt următoarele:

- Aplicația de simulare trebuie să fie intuitivă și ușor de utilizat de către utilizator
- Aplicația de simulare trebuie să afișeze rezultatul simulării într-un timp rezonabil

Diagrama de use-case:



Descrierile de use-case:

- Configurarea simulării:
  - Actor primar: utilizatorul
  - Principalul scenariu de succes:
    1) Utilizatorul introduce valorile pentru: numărul de clienți, număr de cozi, intervalul de simulare, ora minimă și ora maximă de sosire și de servire
    2) Utilizatorul face clic pe butonul de “run” a datelor introduse

3) Aplicația validează datele și, dacă acestea sunt corecte, începe simularea
- Secvență alternativă -> Valori invalide pentru parametrii de configurare:
  1) Utilizatorul introduce valori invalide pentru parametrii configurării simulării
  2) Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă valori valide
  3) Scenariul revine la pasul 1

➢ Pornirea simulării:
- Actor primar: utilizatorul
- Principalul scenariu de succes:
  1) Utilizatorul apasă pe butonul de 'run'
  2) Aplicația afișază, atât într-un fișier .txt, cât și într-o interfață real-time, secunda de când a început simularea, clienții care așteaptă și cozile cu clienți sau fără
  3) Aplicația afișază, de asemenea, timpul mediu de așteptare, timpul mediu de servire și ora la care au fost cei mai mulți clienți în așteptare în cozi
- Secvență alternativă -> Valori invalide pentru parametrii de configurare:
  1) Utilizatorul introduce valori invalide pentru parametrii configurării simulării
  2) Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă valori valide
  3) Scenariul revine la pasul 1

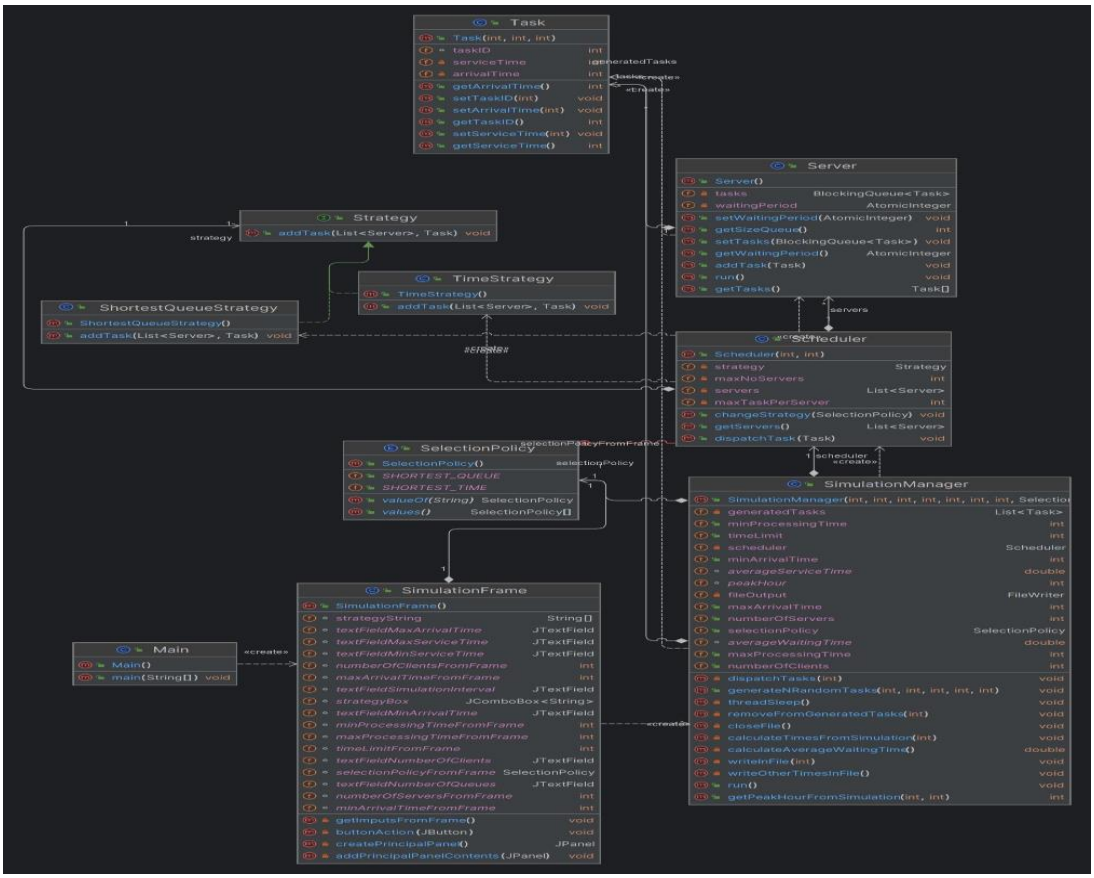
3. Proiectare

1) Proiectarea OOP

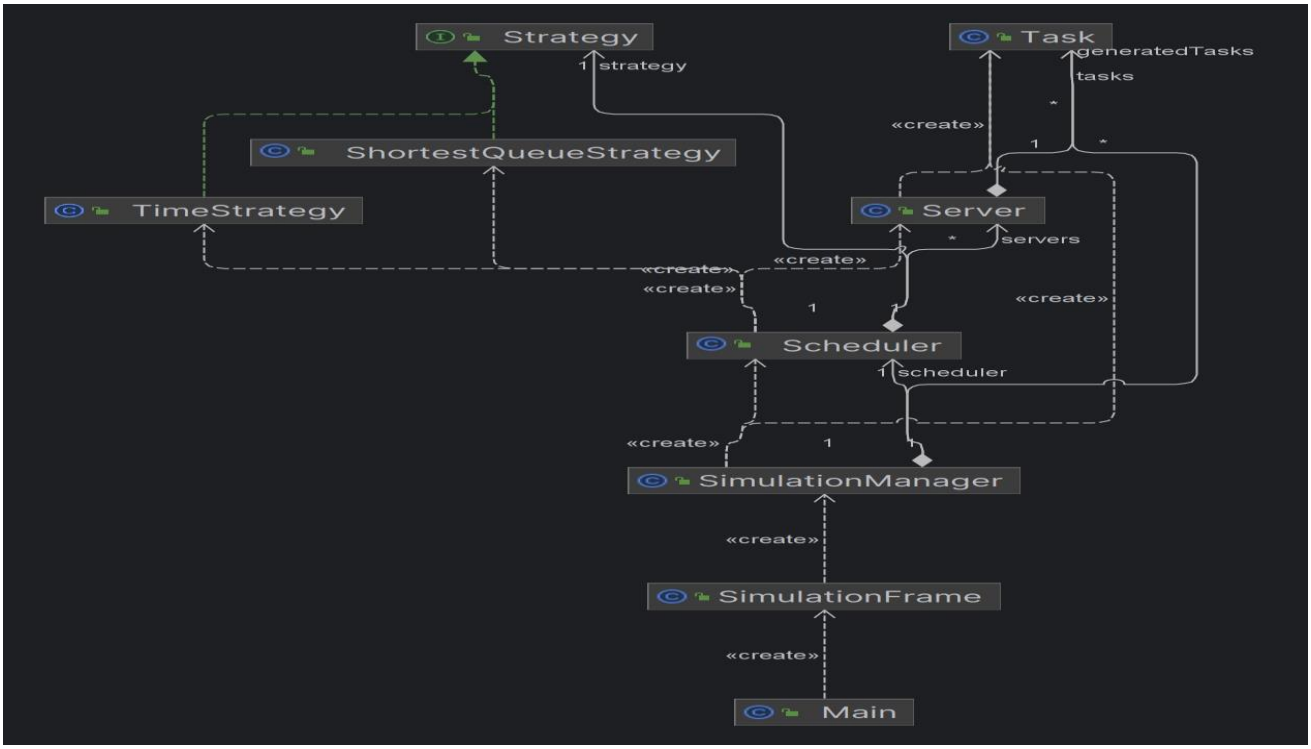
În proiectarea aplicației noastre de simulare folosind OOP, am identificat principalele entități și concepte din domeniul problemei noastre. Acestea includ clienții, cozile, timpul de simulare, intervalele de timp pentru sosire și servire și interacțiunea cu utilizatorul. Am organizat aceste entități în clase corespunzătoare pentru a gestiona funcționalitățile și datele asociate.

2) Diagramele UML de clase și de pachete

Clase:



Pachete:



3) Structurile de date folosite
Structura de date de tip AtomicInteger: asigură că modificările asupra variabilei sunt efectuate în mod atomic, ceea ce înseamnă că modificările sunt efectuate ca o singură operație nedivizibilă și nu pot fi intrerupte de alte fire de execuție.
Structura de date de tip BlockingQueue: oferă metode care permit adăugarea și eliminarea de elemente din coadă, precum și operațiuni care pot bloca firul de execuție până când se îndeplinește o anumită condiție, cum ar fi adăugarea unui element într-o coadă completă sau eliminarea unui element dintr-o coadă goală.

4) Interfețe definite:
Interfața `SimulationFrame`: utilizează biblioteca Swing pentru a crea și gestiona componente precum ferestre, panouri și butoane. Această interfață conține următoarele:
- 7 TextField-uri, pentru a introduce date pentru simulare
- 1 ComboBox, pentru a alege strategia pe care vrem să o utilizăm
- 1 buton, pentru a porni simularea
- metode pentru gestionarea acțiunilor butonului, inclusiv validarea numerelor introduse
- clase din alte pachete(Server, Scheduler, etc.)

5) Algoritmi folosiți:
Algoritmi pentru operații matematice: am implementat algoritmi pentru strategia ‘Shortest Queue’ și ‘Shortest Time’, dar și pentru restul metodelor pe care le utilizez în aplicație.

4. Implementare

a) Clasa `SimulationManager`:

Câmpuri:
1. `timeLimit`: un câmp public pentru timpul de simulare
2. `numberOfClients`: un câmp public pentru numărul de clienți

3. ‘numberOfServers’: un câmp public pentru numărul de cozi
4. ‘minArrivalTime și maxArrivalTime’: câmpuri publice pentru intervalul timpurilor de sosire
5. ‘minProcessingTime și maxProcessingTime’: câmpuri publice pentru intervalul timpurilor de servire
6. ‘selectionPolicy’: câmp public pentru setarea strategiei
7. ‘fileOutput’: câmp privat pentru un fișier

Metode importante:

1. `generateNRandomTasks()`: generează N task-uri random în funcție de numarul de clienți, timpii de sosire și de servire
2. `run`: metodă care pornește thread-ul pentru execuție

b) Clasa `Scheduler`

Câmpuri:

1. `servers`: O listă care stochează numărul de servere
2. ‘maxNoServers’: numărul maxim de servere
3. ‘maxTaskPerServer’: numărul maxim de task-ul pentru fiecare server în parte

Metode importante:

1. `changeStrategy(SelectionPolicy policy)`: setează strategia în funcție de valoarea parametrului dat ca și argument
2. `dispatchTask(Task t)`: adaugă un task nou în funcție de strategia aleasă

c) Clasa `Server`

Câmpuri:

1. ‘tasks’: un BlockingQueue care simulează o coadă a unui task
2. ‘waitingPeriod’: un AtomicInteger care reține valoarea de așteptare pentru fiecare client în parte

Metode importante:

1. `addTask(Task t) `: adaugă un nou task și adună valoarea de servire la waitingPeriod
2. `run`: metodă care pornește thread-ul pentru execuție

d) Clasa `Task`

Câmpuri:

1. ‘arrivalTime’: timpul de așteptare al unui task
2. ’serviceTime’: timpul de servire al unui task

Metode importante:

1. getters pentru fiecare câmp
2. setters pentru fiecare câmp

e) Clasa ShortestQueueStrategy:

Câmpuri: -

Metode: addTask(Task t): adaugă un nou task în server, în funcție de dimensiunea celei mai mici cozi

f) Clasa TimeStrategy:

Câmpuri: -

Metode: addTask(Task t): adaugă un nou task în server, în funcție de valoarea minimă a timpului de servire

g) Clasa ‘SimulationFrame

Câmpuri:

1. 7 textField-uri, pentru fiecare parametru din simulare
2. un JComboBox, pentru alegerea strategiei
3. un buton ‘run’

Metode importante:

1. ‘createPrincipalPanel’: creează panoul principal
2. ‘addPrincipalPanelContents(JPanel panel)’: adaugă câmpurile în panoul principal
3. ‘buttonAction()’: metoda pentru acționarea butonului ‘run’ din interfață

5. Rezultate

Pentru oricare set de parametrii introduși în interfața grafică, output-ul se va afișa atât într-un fișier .txt, cât și într-o altă interfață în timp real.

De asemenea, avem și 3 fișiere .txt, acolo fiind afișate rezultatele simulării pentru un număr cunoscut de imput-uri.

6. Concluzii

În concluzie, această temă a fost una destul de interesantă, prin prisma faptului că am creat, de la 0, o aplicație care simulează un fel de ‘magazin’, prin clienți, cozi și timpi specifici. De asemenea, utilizarea unei interfețe grafice pentru inserarea argumentelor necesare simulării, respectiv alegerea unei strategii, a fost un lucru interesant de implementat.

Din această temă am învățat cum se poate manageria o aplicație pe thread-uri în Java. De asemenea, am învățat cum să utilizez structura de date de tip BlockingQueue și AtomicInteger, ceea ce o să mă ajute foarte mult în viitorul apropiat.

Niște posibilități de dezvoltări ulterioare ar putea fi:

- Înfrumusețarea interfeței grafice(adăugare culori, font mai frumos, etc.)
- Implementarea unei interfețe care afișază, în timp real, evenimentele
- Alte strategii de manageriere a simulării

7. Bibliografie

1. *https://dsrl.eu/courses/pt/materials/PT_2024_A2_S1.pdf*
2. *https://dsrl.eu/courses/pt/materials/PT_2024_A2_S2.pdf*
3. *https://www.geeksforgeeks.org/atomicinteger-addandget-method-in-java-with-examples/*
4. *https://www.geeksforgeeks.org/blockingqueue-interface-in-java/*