# DOCUMENTAȚIE

TEMA *2*

NUME STUDENT: Vadean Catalin Ioan
GRUPA: 30223

# CUPRINS

# 1. Obiectivul temei

Proiectați și implementați o aplicație al cărei scop este să analizeze sistemele bazate pe cozi (FIFO – First In, First Out) prin simularea a N clienți care sosesc pentru un serviciu, și care se împart în Q cozi. Aceștia își așteaptă rândul, își rezolva problema iar apoi părăsesc coada, respectiv prin calcularea timpului mediu de așteptare, timpului mediu pe care îl petrec în capul cozii și ora de vârf.

Obiectivele secundare:

- Crearea unui număr de Thread-uri egal cu numărul de cozi disponibile
- Fiecare coadă corespunde unui Thread
- Clienții (sau Task-urile) sunt distribuiți în fiecare coadă conform unei politici stabilite
- Împărțirea pe clase corespunzătoare, alegerea structurilor de date potrivite
- Realizarea unor interfețe grafice pentru utilizator, una pentru introducerea datelor și pornirea simulării, una pentru simularea propriu-zisă.
- Respectare conceptelor programării OOP.

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

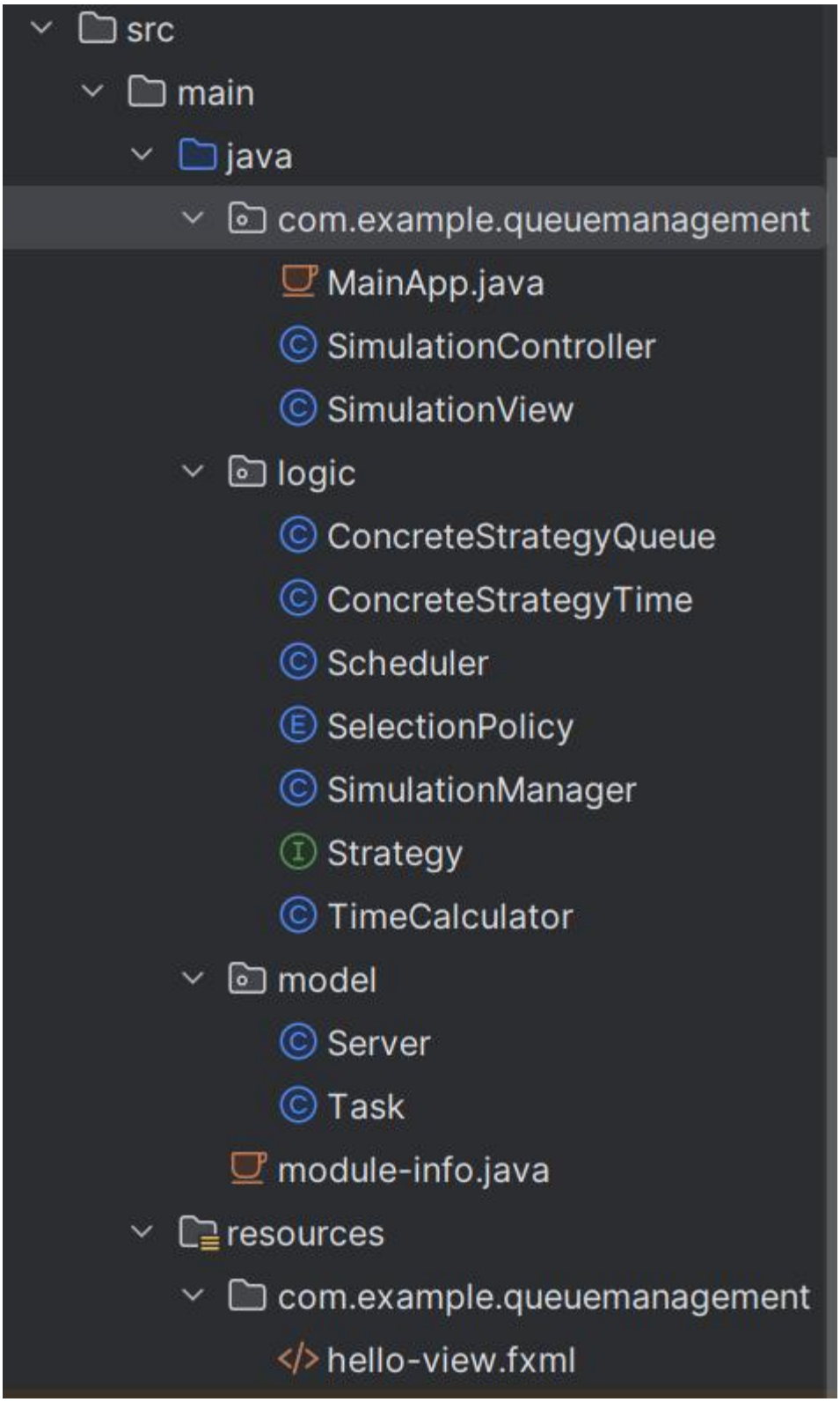
Problema de față are ca date de intrare valorile necesare simulării, și anume numărul de cozi sau servere disponibile, numărul de clienți sau task-uri, timpul simulării adică pe ce perioadă se va întinde simularea, timpii minim și maxim de Arrive, adică timpii între care un client poate intra într-o coadă, timpii minim și maxim de procesare sau de serviciu, adică timpii între care un client va sta în fruntea cozii și își va rezolva problemele la „ghișeu”, respectiv politica de intrare într-o coadă a unui nou client.

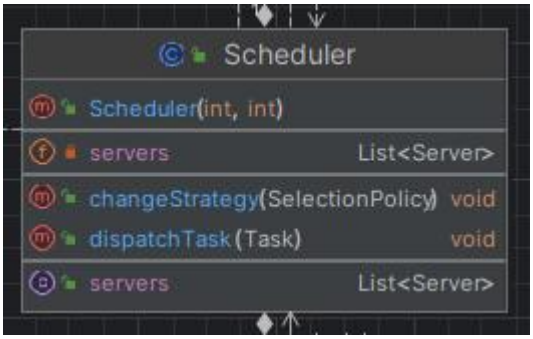
Scenariul de folosire este următorul: utilizatorul introduce în fereastra „SIMULATION SETUP” parametrii descriși anterior, va selecta politica de intrare în coadă și va apăsa butonul SIMULATE. La apăsarea acestuia, fereastra de introducere a datelor de intrare se va închide, și se va porni simularea care va fi prezentată în timp real într-o altă interfață grafică. În același timp, într-un fișier text se vor scrie toate log-urile simulării. Vor fi afișate timpul curent, task-urile în așteptare respectiv cozile, care vor fi închise sau vor avea clienți în așteptare. La finalul timpului de simulare, se va afisa timpul mediu de servire a clientilor.

# 3. Proiectare

Pentru proiectarea aplicatiei m-am folosit de scheletul care a fost pus la dispozitie in prezentarea temei.Pe langa clasele respective, au fost folosite clasele View si SimulationView, care au rolul de a permite introducerea datelor de intrare, respective de a afisa in timp real informatii despre simulare.

De asemenea, clasele au fost structurate pe pachete:


src
main
java
com.example.queuemanagement
MainApp.java
SimulationController
SimulationView
logic
ConcreteStrategyQueue
ConcreteStrategyTime
Scheduler
SelectionPolicy
SimulationManager
Strategy
TimeCalculator
model
Server
Task
module-info.java
resources
com.example.queuemanagement
hello-view.fxml

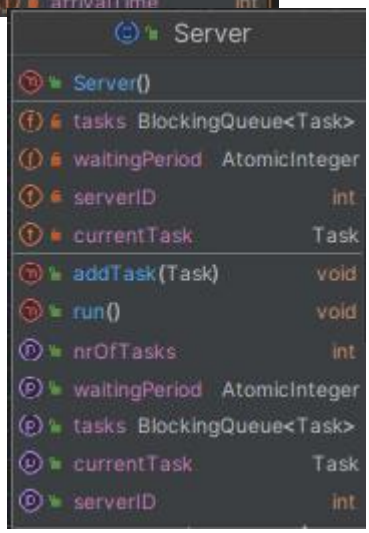



Interfata grafica, compusa din:

# 4. Implementare

Clasa **Task** simbolizează clienții ce
constructor care instanțiază un Clien
arrivalTime și un serviceTime, care su






În enum-ul **SelectionPolicy** sunt „hard-codate" tipurile de politica de adăugare în coadă, care vor fi descrise fiecare în parte: SHORTEST_QUEUE, SHORTEST_TIME.

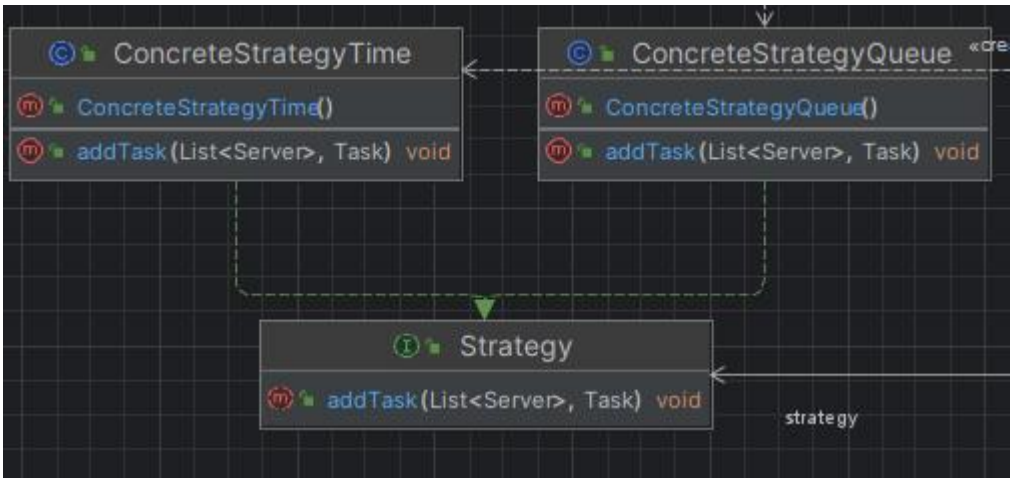
C
si clientii asteapta. Pentru a usura afisarea, am introdus un camp numit serverID, si un atribut de tip Task, in care va fi stocat Task-ul current, aflat in capul cozii.

Pentru a adauga elemente de tip Task in coada, folosim metoda addTask, care va adauga un Task primit ca parametru in coada de tip BlockingQueue. Timpului de asteptare al cozii i se va adauga valoarea timpului de procesare al Task-ului adaugat.

Interfata Strategy contine metoda addTask, care este implementata de clasa ConcreteStrategyTime, respectiv ConcreteStrategyQueue. Clasa ConcreteStrategyQueue primeste ca parametru o lista de servere si un Task, si adauga Task-ul in serverul cu cele mai putine Task-uri.

Clasa ConcreteStrategyTime primeste ca parametru o lista de servere si un Task, adaugand Task-ul in serverul cu cel mai scurt WaitingPeriod.

Clasa Scheduler contine o lista de servere sau cozi, numarul maxim de servere, numarul de clienti si un atribut de tip Strategy. Rolul acestei clasa este de a adauga in servere task-uri conform strategiei alese, si de a lansa spre executie serverele.



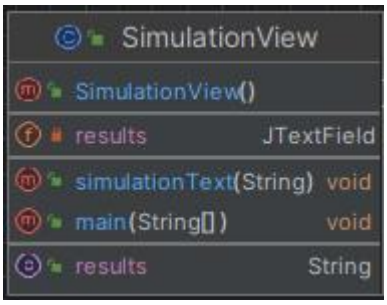


Clasa View reprezinta primul contact al utilizatorului cu aplicatia, prin aceasta introducand datele de intrare pentru simulare.


Clasa SimulationManager reprezinta controlul intregii aplicatii. Aceasta cuprinde atributele corespunzatoare datelor de intrare, care sunt primite ca parametru si luate din interfata grafica de introducere a datelor. Constructorul clasei initializeaza variabilele pentru simulare, apoi se genereaza sarcinile (‘Task’) aleatoriu, folosind metoda “generateNRandomTasks()”. Apoi, se implementeaza logica de executie a thread-ului. Intr-o bucla care se repeta pana la

atingerea limitei de timp, sarcinile care trebuie sa inceapa in momentul curent sunt procesate. Sarcinile sunt trimise catre servere prin intermediul clasei Scheduler, care utilizeaza o politica de selectie specifica. Apoi, informatiile sunt transmise si afisate in interfata grafica SimulationView.

Prin SimulationView, se afiseaza datele in timpul real, task-urile de asteptare si cozile cu task-urile lor, alaturi de un TextField ce afiseaza Average Service Time-ul. La inchiderea ferestrei, se va termina si executia programului.




## 5. Rezultate

Pentru testare, s-au folosit atat date de intrare aleatorii, cat si cele propuse in cerinta temei.

In cadrul primului test, care presupunea ca date de intrare un numar de 4 clienti, 2 cozi, un timp de simulare de 60 de secunde, arrivalTime-ul cuprins intre 2 si 30 de secunde si timpii de procesare de la 2 pana la 4 secunde, nu s-a atins timpul de simulare maxim, simularea oprindu-se mai repede din cauza numarului mic de clienti.

In cadrul celui de-al doilea test, numarul de client era 50, ce trebuiau repartizati intr-un numar de 5 cozi, in timpul de simulare de 60 de secunde, cu arrivalTime-ul cuprins intre currentTime-ul de 2 si 40, cu timpul de procesare de la 1 la 7. Depinzand de task-urile generate si de arrivalTime-urile lor, se observa doua scenarii. Unul in care toate task-urile sunt finalizate in timp util, adica in timpul de simulare. Al doilea scenario presupune terminarea simularii in conditiile in care inca exista taskuri in executie, in cozi.

In cadrul celui de-al treilea test, datele de intrare puse la dipozitie sunt un numar de 1000 de Task-uri, 20 de cozi in care vor fi repartizare, un timp de simulare de 200 de secunde, timpii de intrare in cozi intre 10 si 100 si timpii de procesare intre 3 si 9. La fiecare rulare a aplicatiei pentru aceste date de intrare, la finalul simularii ramanea un numar semnificativ de task-uri in cozi, care nu au fost finalizate.

## 6. Concluzii

Prin aceasta tema consider ca am aprofundat notiunea de fir de executie, cu care am interaactionat foarte putin inainte de inceperea acestei teme respective importanta organizaraii si proiectarii corespunzatoare a claselor, fapt ce usureaza considerabil implementarea.

# 7. Bibliografie

https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_2.pdf

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

https://www.w3schools.com/java/java_threads.asp