Universitatea Tehnică din Cluj-Napoca

Facultatea de Automatică și Calculatoare

Secția Calculatoare

Disciplina Tehnici de Programare



Simulator de Cozi

2020-2021

Student: Balog Helga

Grupa: 30227

Profesor Coordonator: Dan Mitrea

Cuprins

[1.Cerințe Funcționale 2](#_Toc1777604167)

[2.Obiective 2](#_Toc257568816)

[Obiectivul principal al acestui proiect este de a citi datele necesare prin interfața grafică și de a afișa pe tot parcursul simulării clienții care se află la cozi și, respectiv, pe cei care încă sunt la cumpărături. 2](#_Toc103820216)

[2.2 Obiective Secundare 2](#_Toc344761372)

[3.Analiza Problemei 3](#_Toc296745694)

[4.Proiectare și Implementare 3](#_Toc905248821)

[4.2 Pachete și Clase 4](#_Toc392747657)

[5.Funționare 7](#_Toc1657620095)

[6.Testare 8](#_Toc2041400534)

[6.Concluzii si Dezvoltări Ulterioare 9](#_Toc1589508386)

[7.Bibliografie 10](#_Toc884033471)

# 1.Cerințe Funcționale

Proiectați și implementați o aplicație de simulare cu scopul de a analiza sistemele bazate pe cozi pentru a determina și minimiza timpul de așteptare al clienților. Aplicația trebuie să citească numărul de clienți, numărul de cozi, timpul maxim de simulare, intervalul în care clienții pot ajunge la cozi și intervalul de timp în care o să fie serviți. De asemenea, simularea trebuie să fie reprezentată pe o altă interfață în timp real. Clienții o să fie generați aleatoriu.

# 2.Obiective

2.1 Obiectivul Principal

Obiectivul principal al acestui proiect este de a citi datele necesare prin interfața grafică și de a afișa pe tot parcursul simulării clienții care se află la cozi și, respectiv, pe cei care încă sunt la cumpărături.

## 2.2 Obiective Secundare

Pentru a atinge scopul acestei aplicații trebuie să urmărim anumite obiective secundare.

În primul rând trebuie să creăm o interfață grafică care să lucreze cu utilizatorul. Acesta trebuie să introducă numărul de clienți, numărul de cozi, numărul maxim de simulare, intervalul în care pot sosi clienții la cozi și intervalul de timp în care aceștia pot fi serviți, ca mai apoi interfața să transmită mai departe aceste date la SimulationManager.

În al doilea rând trebuie să generăm aleatoriu n clienți și să creăm q cozi. Pentru fiecare coadă o să partajăm anumiți clienți, în funcție de cât de mulți oameni se află momentan la acea coadă.

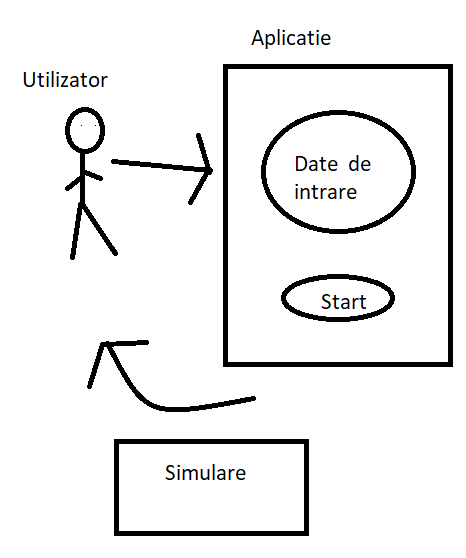
În al treilea rând, aplicația trebuie să afișeze pe o altă interfață grafică, simularea în timp real.

# 3.Analiza Problemei

Această aplicație susține următoarea scenă de utilizare: un utilizator introduce datele de simulare și apasă pe butonul de start. În momentul în care s-a apăsat butonul de start, începe simularea pe o alta interfață grafică. Pe această interfață se pot observa clienții care sunt la cozi, clienții care încă fac cumpărături și timpul în secunde.

De asemenea, interfața va afișa media timpul de servire și ora de vârf.

Mai jos se poate observa o diagrama UseCase.



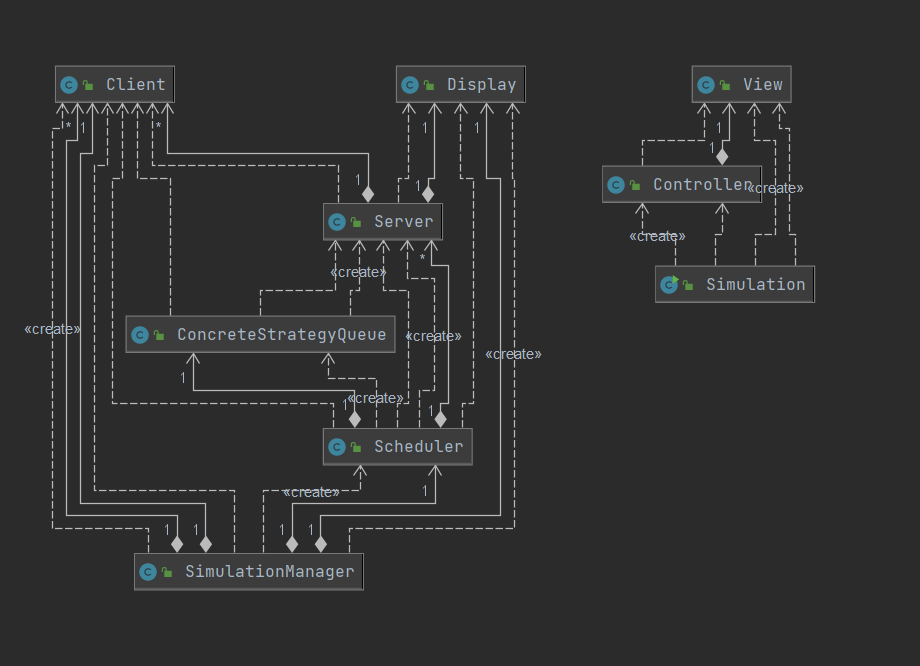
# 4.Proiectare și Implementare

4.1 Proiectare

Aplicația este implementată în limbajul de programare Java și are următoarele pachete: view, controller, client și simulationManager. Pachetul view se ocupă cu crearea și afișarea interfeței grafice. Pachetul controller detectează când se apasă butonul de start, iar în acel moment pornește SimulationManager. Pachetul simulationManager conține mai multe clase, care se ocupă cu crearea threadurilor, cu împărțirea clienților la cozi și cu afișarea simulării pe o altă interfață.

Pentru a menține siguranța threadurilor, am folosit BlockinDequeue, pentru memorarea clienților.

În imaginea de mai jos se poate observa o diagramă UML a aplicației.



## 4.2 Pachete și Clase

Pachetul View

Pachetul View, conține două clase, clasa View și clasa Interval. Clasa view se ocupă de realizarea interfeței. Aceasta are mai multe instanțe importate din pachetul javax.swing.\*

Clasa Interval are scopul de a ne ajuta la citirea intervalului de timp în care sosesc clienții la casă, respectiv intervalul în care aceștia pot fi serviți și la transmiterea datelor mai departe în SimulationManager, unde ne vom folosi de aceste intervale pentru generarea clienților.

Pachetul Client

Pachetul client conține o singură clasă intitulată la fel, care este compusă din trei instanțe: ID, tArrivel, tService. Astfel un client va fi reprezentat de aceste trei câmpuri.

Pachetul Controller

Pachetul controller, conține o singură clasă intitulată la fel. Această clasă așteaptă apăsarea butonului de start din prima interfață. Când acest buton este apăsat, datele de intrare sunt citite și date mai departe la SimulationManager.

Pachetul simulationManager

Pachetul simulationManager conține mai multe clase. Prima clasă, intitulată SimulationManager primește datele de la intrare și generează n clienți. Această clasă extinde clasa Runnable, deci implementează și metoda run(). Anterior, în clasa Controller, după ce s-a instanțiat obiectul de tipul SimulationManager, s-a creat un thread cu acesta. Astfel, acel thread va executa metoda run() din clasa SimulationMaager .

Metoda run() are un întreg currentTime și cât timp acel currentTime este mai mic sau egal cu timpul maxim de simulare va afișa în cea de-a doua interfață grafică timpul. Dacă găsește un client cu timpul de sosire egal cu timpul curent îl trimite mai departe pentru a-l adăuga la o coadă. De asemenea va afișa și clienții care încă nu au ajuns la cozi. După aceea threadul va fi oprit o secundă. Pe lângă interfață, SimulationManager scrie timpul și într-un fișier.txt.

Următoarea clasă este Scheduler. În constructorul acestei clase se creează n servere după care și threaduri cu aceste servere. De fapt, acest constructor creează q cozi.

Clasa Server reprezintă cozile. La fel ca și SimulationManager, Server extinde clasa Runnable, deci implementează și metoda run(). Prin metoda run() se afișează pe interfața grafică cozile cu clienții de la acestea și, la fel, se scrie și în fișierul.txt. În același timp, la fiecare executare a acestei metode, se decrementează timpul de servire al primului client. Astfel dacă timpul de servire ajunge să fie 0, se scoate acest client din coadă.

Clasa ConcreteStrategyQueue adaugă un client la coada care are cel mai mic număr de clienți, astfel clientul alege coada la care trebuie să aștepte cel mai puțin.

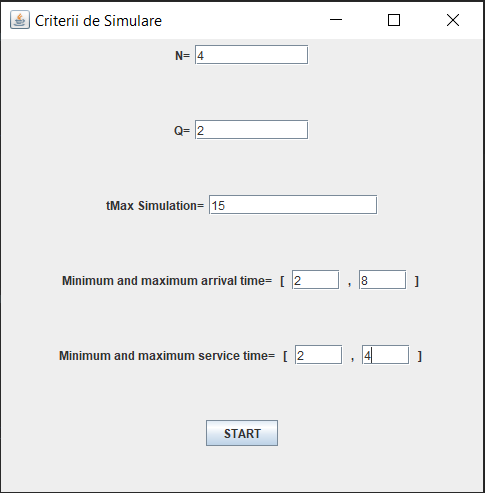
Clasa Display este la fel ca și clasa View din pachetul view. Aceasta extinde JFrame și construiește interfața grafică pe care se va putea vizualiza simularea în timp real. Display are mai multe metode care sunt apelata pe tot parcursul simulării, cum ar fi setTime, setQueue, setMedie... etc. Aceste metode nu fac altceva decât să actualizeze interfața de simulare.

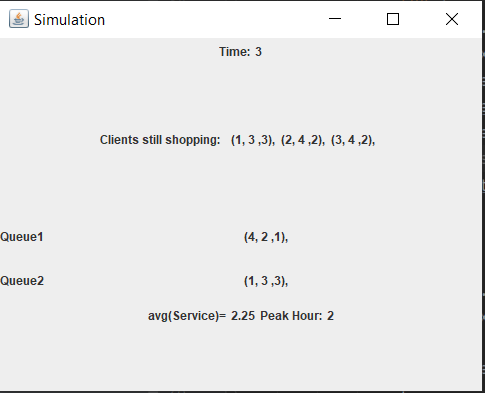
Acest pachet conține de asemenea o interfață, numită Strategy, care are o singură metodă abstractă addTask. Clasele care conțin strategia de împărțire a clienților la cozi (ex ConcreteStrategyQueue) implementează această clasă. La dezvoltări ulterioare, această interfață ar putea fi implementată de mai multe strategii.

Pachetul simulation

Pachetul simulaton, conține o singură clasă, intitulată la fel. Această clasă inițializează un view și un controller și setează vizibilitatea interfeței la true.

În imaginea de mai jos se poate observa interfața grafică pentru citirea datelor și cea pentru afișarea simulării.





# 5.Funționare

În prima fază , utilizatorul introduce datele de simulare, de exemplu:

N=4

Q=2

Simulation=30

Minimum and Maximum arrivel time= [2, 20]

Minimum and Maximim service time=[2, 5]

Și apasă pe butonul de start.

În acel moment Controller-ul citește datele de intrare, inițializează un SimulationManager cu acele date și pornește un thread cu obiectul de tipul SimulationManager. În momentul în care am inițializat simulationManager, în constructului acestuia se va inițializa și un Scheduler. Acest scheduler va crea q servere și threaduri pentru fiecare server. Între timp, interfața pentru simularea va fi creată și afișată. Threadul pornit cu obiectul de tipul SimulationManager va executa metoda run(), adică va afișa timpul în secunde pe interfață și în fișier. De asemenea, dacă găsește client cu timpul de sosire egal cu timpul curent, îl scoate din lista de clienți și îl trimite mai departe pentru a fi adăugat la o coadă. Celelalte threaduri, care se ocupă de cozi, verifică daca îs clienți la coadă sau nu. Dacă da, îi afișează pe interfață și în fișierul text, și scade cu o unitate timpul de servire al primului client. Toate acestea se execută până când currentTime ajunge să fie mai mare decât timpul maxim de simulare.

Pe interfața de simulare, la final, se va afișa și timpul mediu de servire și ora de vârf. De asemenea , cozile care au listele de clienți goale, adică nu au niciun client, o să afișeze textul ‘Free’ . La fel, când nu a mai rămas niciun client în magazin, la lista de clienți se va afișa textul ‘No one is shopping right now’.

Scrierea în Fișierul Text

Scrierea în fișier are loc datorită fileWriter-ului declarat în constructorul clasei SimulationManager. Scrierea se face utilizând metoda write(). De fiecare dată când scriem într-un fișier codul trebuie înconjurat de comenzile try și catch pentru a detecta și pentru a prinde orice excepție care s-ar putea să apară în momentul scrierii în fișier. Dacă se detectează o excepție, se va afișa în terminal textul ‘Probleme cu fileWriter’ și se va continua execuția programului.

Când inițializăm fileWriter-ul în dăm ca și parametru numele unui fișier text. Dacă acesta există deja, se va înlocui conținutul fișierului cu informații noi, iar dacă nu există fișierul atunci îil creează.

# 6.Testare

Am testat aplicația de trei ori, cu date de intrare diferite.

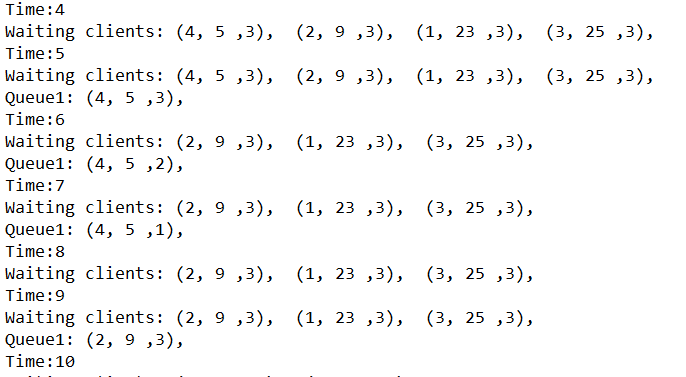
Primul test are următoarele date de intrare: n=4, q=2, simulation=60, minimum and maximm arrivel time= [ 2, 30] și minimum and maximum service time= [2, 4]. Aplicația va genera aleatoriu patru clienți și va porni două threaduri pentru aceștia. Fiecare client va avea timpul de sosire între 2 și 30, iar timpul de servire între 2 și 4. În acest caz, timpul se simulare va fi de 60 de secunde.

Aplicația a funcționat într-un mod eficient și corect. În prima imagine de mai jos se poate observa o porțiune din fișierul text în care s-au printat acțiunile.

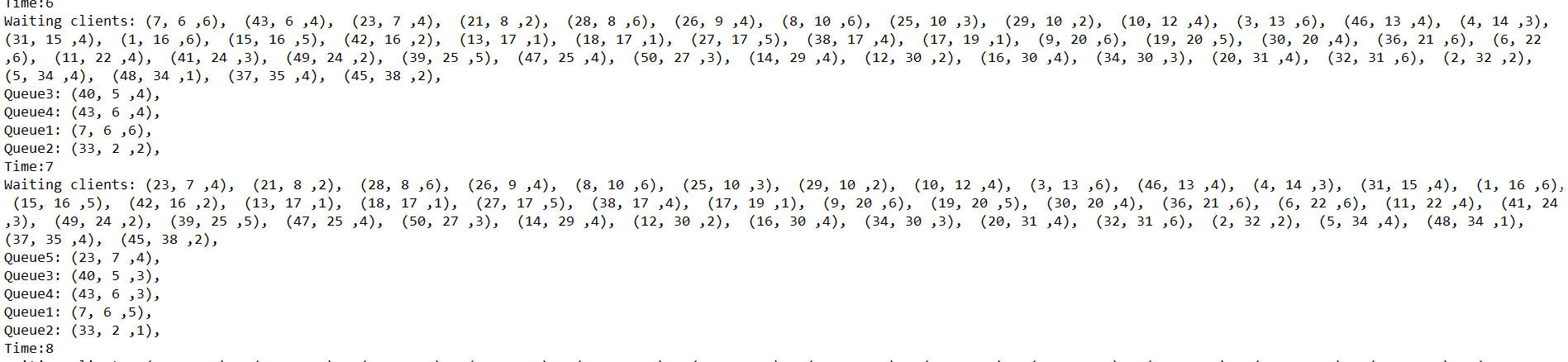
Al doilea test are următoarele date de intrare: n=50 , q=5, simulation=60, minimum and maximum arrivel time=[2, 40] , minimum and maximim service time =[1, 7]. Aplicația va genera aleatoriu 50 clienți și va porni 5 threaduri pentru aceștia. Fiecare client va avea timpul de sosire între 2 și 40, iar timpul de servire între 1 și 7. În acest caz, timpul se simulare va fi de 60 de secunde.

Aplicația a funcționat corect și eficient. În a doua imagine puteți observa o porțiune din fișierul text care s-a generat.

După cum se poate observa, se afișează doar cozile care nu au lista de clienți goală.



*Imaginea 1*



*Imaginea 2*

Al treilea test are următoarele date de intrare: n=1000, q=20, simulation=200, minimum and maximum arrivel time=[10, 100], minimum and maximum serive time=[3, 9]. Aplicația va genera aleatoriu 1000 clienți și va porni 20 de threaduri pentru aceștia. Fiecare client va avea timpul de sosire între 10 și 100, iar timpul de servire între 3 și 9. În acest caz, timpul se simulare va fi de 200 de secunde.

În interfața grafică aplicația funcționează corect și eficient , dar la scrierea în fișier anumite cozi se intersectează. De asemenea, în fișier se afișează, la fiecare secundă, clienții care încă nu au ajuns la cozi, astfel fișierul este destul de stufos. (la testul 3)

# 6.Concluzii si Dezvoltări Ulterioare

În concluzie, simulatorul de cozi funcționează eficient și corect și își atinge obiectivul principal de a simula, într-un timp mult mai rapid, procesul de împărțire al clienților la cozi și de plată la un magazin. Clienții sunt serviți în timpul specificat în instanța tServcie și sunt împărțiți la cozi într-un mod rapid și eficient. De asemenea, timpul de simulare este respectat și afișat în simulator și în fișierul text într-un mod corect.

Ca și dezvoltare ulterioară s-ar putea actualiza codul astfel încât threadurile să nu afișeze în fișierul text în același timp conținutul cozilor. De asemenea s-ar putea introduce o altă strategie de împărțire a clienților la cozi. În aplicație, clienții noi sunt puși la cozile care sunt cele mai scurte. S-ar putea face o strategie care să pună clienții la cozile care au cel mai rapid angajat și care nu are încă numărul maxim de clienți la coadă atins.

Din punct de vedere estetic, am putea considera că magazinul are o capacitate maximă de 50 de clienți. Clienții care s-au generat în plus o să intre în magazin, doar după ce scade numărul

actual de clienți în magazin. De exemplu: în momentul în care primul client a fost servit, poate intra următorul client. Astfel la afișarea pe interfață și în fișier nu o să se aglomereze clienții.

Tot din punct de vedere estetic, s-ar putea îmbunătăți cele două interfețe. Prima ar putea avea teste de validare și de avertizare în cazul în care utilizatorul introduce date greșite. A doua interfață ar putea pune în loc de Queue + numărul cozii, o imagine cu o casă de marcat, și în loc de datele clientului ( id, tArrivel, tService) ar putea pune un punct sau alt obiect prin care să reprezinte clientul.

# 7.Bibliografie

1. <https://www.youtube.com/watch?v=TCd8QIS-2KI> (04.04.2021)

2. <https://www.geeksforgeeks.org/generating-random-numbers-in-java/> (27.03.2021)

3. <http://users.utcluj.ro/~igiosan/teaching_poo.html> (04.04.2021)

4. <https://www.w3schools.com/java/java_files_create.asp> (06.04.2021)

5. <https://www.tutorialspoint.com/java/java_filewriter_class.htm> (06.06.2021)