SIMULATOR DE COZI

Mihaila Gabriel

grupa 30229

UTCN

Prof indrumator: Marcel Antal

**Cuprins**

1. Assignment objective
   1. Main objective
   2. Secondary object
2. Problem analysis and use case diagram
   1. Problem analysis
   2. Use case diagram
3. Design
   1. GUI
   2. Design decisions
   3. Packages
   4. Data structures
   5. UML Diagram
4. Implementation
5. Results
6. Conclusions
7. Bibliography

1. Assignment objective

1.1.Main objective

Obiectivul principal al acestui proiect este de a construi, asa cum se mentioneaza si numele, un simulator de coada pentru un numar de clienti care asteapta sa fie deserviti la un anumit serviciu, cum ar fi un supermarket. Aplicatia ar trebui sa poata interactiona cu utilizatorul prin intermediul unei interfate grafice care contine numarul de clienti, numarul de cozi, timpul de sosire minim si maxim pentru clienti si timpul de serviciu minim si maxim. Pe baza acestor date, aplicatia ar trebui sa poata simula serviciul clientilor la cozi, afisand la fiecare pas de simulare (timpul de simulare) ce clienti asteapta, care sunt deserviti si care a fost timpul mediu de serviciu. Toate aceste date trebuie afisate intr-un fisier de iesire. Aplicatia necesita interactiune cu utilizatorul, pe langa fisierul de intrare.

1.2.Secondary objective

Aceste obiectiv e legat mai mult de cerintele tehnice ale implementarii si de abilitatile cerute de mine ca student sa le folosesc, mai degraba decat de caracteristicile abstracte ale aplicatiei. Daca nu se specifica altfel, realizarea acestor obiective poate fi observata in capitolul 4 (implementare). Obiectivele secundare sunt urmatoarele:

* + 1. Have an Object-Oriented Programming design;
    2. Have classes with maximum 300 lines;
    3. Have methods with maximum 30 lines;
    4. Respect the Java naming conventions (Camel case);
    5. Have a quality documentation;
    6. Random Client Generator
    7. Multithreading: one thread per queue
    8. One test run and saved: in-test-1.txt
    9. .jar file uploaded and configured to run according to deliverable requirement
    10. Appropriate synchronized data structures to assure thread safety (avoid synchronized keyword as much as possible)
    11. Queues should open/close dynamically. Initially all queues are closed. When clients are distributed to the queues, they become open as needed. When a queue becomes empty, it is closed, and the corresponding thread is paused
    12. Other two tests given in requirements (in-test-2.txt, in-test-3.txt)
    13. Compute average waiting time
    14. The application should permit to be run with the following command: java-jar; PT2020\_30421\_Chirtos\_PetruCristian\_Assignment\_2.jar in.txt out.txt; Remark: the files should be in the same directory as the jar file in order for the application to work properly.

1. Problem analysis and use case diagram
   1. Problem analysis

Problema necesita crearea unei aplicatii pentru simularea unui set de clienti si cozi, fiecare client avand un anumit timp de sosire si un anumit timp de serviciu. Fiecare coada ar trebui sa fie reprezentata de un fir separat, deci este necesara aplicarea de multithreading, iar siguranta thread-ului este o necesitate. Initial, toate firele sunt inchise. Odata ce un client isi introduce coada, firul incepe si este intrerupt cand nu mai are clienti.

Aplicatia ar trebui sa poata face urmatoarele:

• Cititi datelor de pe interfata grafica InputView;

• Generrarea random datelor pe baza constantelor citite de pe interfata grafica InputView;

• Pornirea simularii si inceperea tipariri rezultatului in fisierele de iesire, precum si pe interfata grafica OutputView;

• Finalizarea simularii cu o iesire corecta si valida

Intrarea are urmatorul format si semnificatie:

N - numarul de clienti;

Q - numarul de cozi;

Simulare T max - valoarea maxima a timpului pentru simulare;

T min sosire, T max sosire - valorile minime si maxime ale timpului in care un client poate sosi;

Serviciu T min, serviciu T max - valorile minime si maxime de timp pe care un client le poate avea in timp ce se afla in coada.

Pentru a rezolva aceste sarcini, a fost necesara o intelegere clara a firelor si a programarii concurente, iar cunostintele au fost obtinute din tutoriale de la primele linkuri din bibliografie, cele mai multe informatii provenind din documentatia oficiala java.

2.2Use case diagram

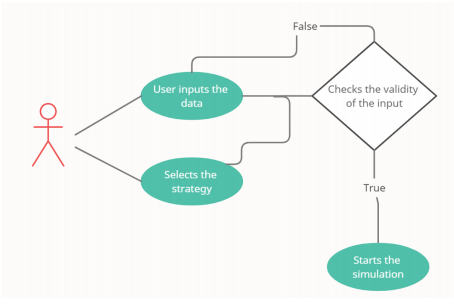
Cazul de utilizare pentru aceasta aplicatie este specific, deoarece citeste limitele, isi genereaza datele si incepe simularea pe baza acestor date. Avand in vedere acest fapt, un scenariu comun pentru utilizarea acestei aplicatii ar arata cam asa:

• Actorul deschide interfata grafica care contine limitele pe care el sau ea doreste sa le ia in considerare pentru simulare;

• Actorul executa comanda si da valori prin intermediul interfetei grafice.

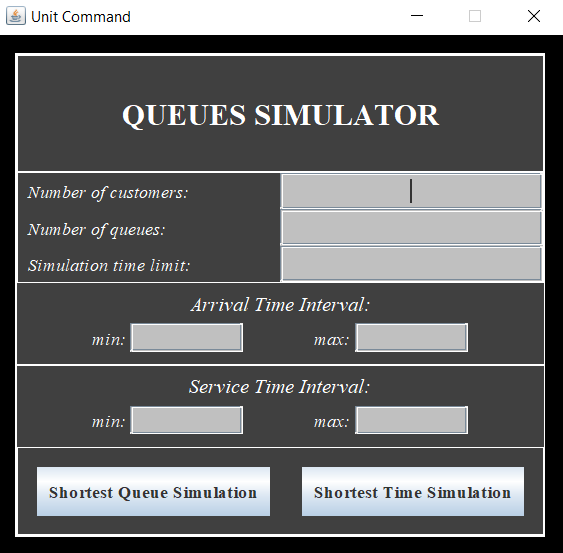
• Daca actorul a selectat limite mari, el sau ea ar trebui sa fie constient de faptul ca ar putea fi necesar un timp de asteptare mic pentru ca iesirea completa sa fie gata;

• Actorul vizualizeaza fisierul de iesire reprezentand rezultatele simularii.



1. Design
   1. InputView

Folosit pentru a furniza intrarea pentru simulare si pentru a porni, de asemenea simulare facand clic pe „butonul de Start simulare”, in functie de strategia aleasa. Este implementat folosind urmatoarele JTextFields si JLabels corespunzatoare.



number of customers -> pentru numarul de clienti.

number of queues -> pentru numarul de servere.

simulation time limit -> pentru durata maxima a simularii.

Arrival Time -> pentru limitele orei de sosire.

Service Time -> pentru limitele timpului de procesare.

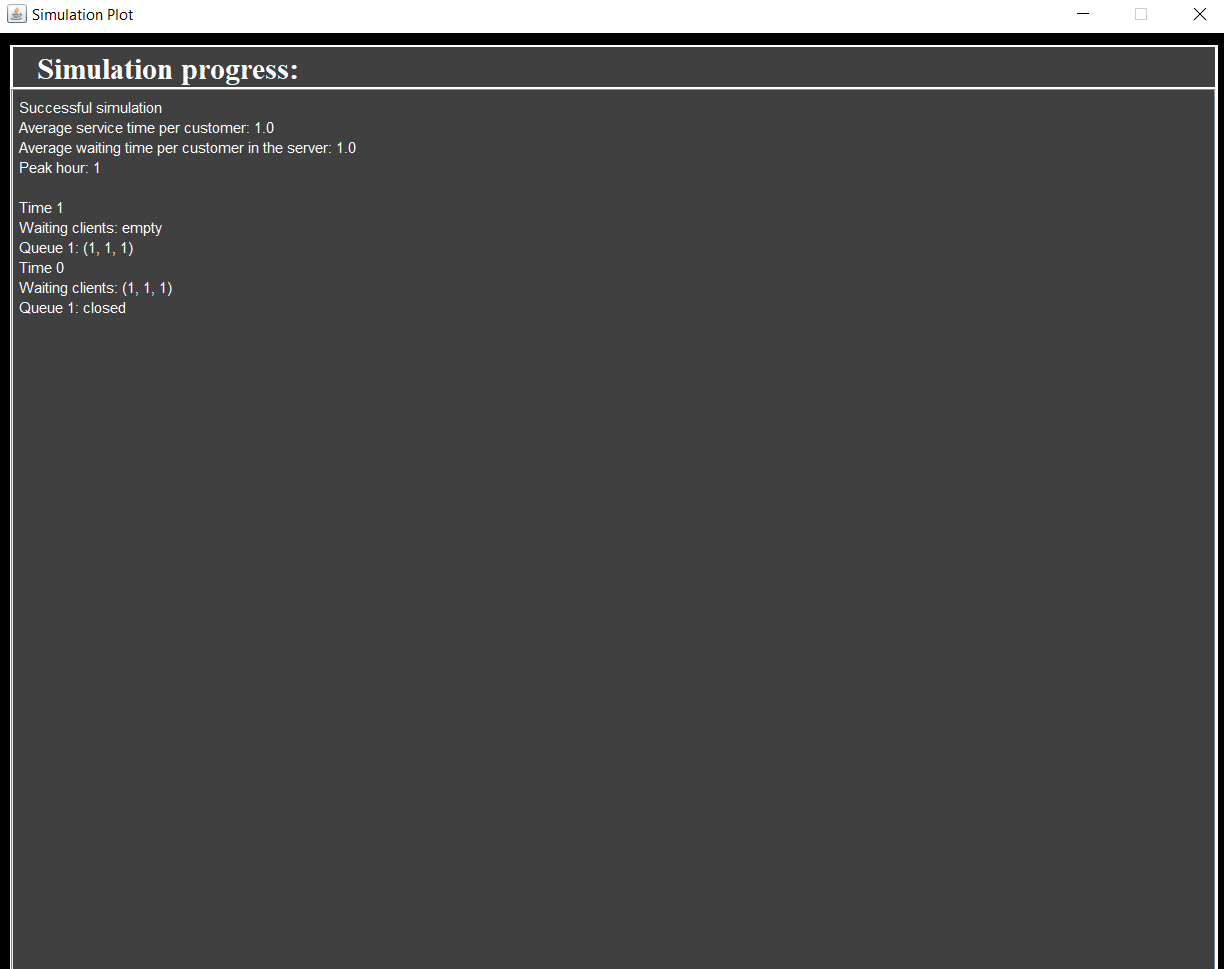
De asemenea, sunt implemntatea doua butoane, "shortestTimeStrategyButton" si „shortestQueueStrategyButton” care folosesc datele si porneste un nou JFrame cu simularea, in functie de strategia aleasa.

OutputView

Folosit pentru a afisa evolutia in timp real a simularii intr-o maniera de defilare, oferind in acelasi timp posibilitatea de a vedea in timp real fiecare modificare in functie de timpul curent al simularii.

Datele sunt afisate de sus in jos, ceea ce inseamna ca orele au aparut in ordine inversa astfel incat utilizatorul nu trebuie sa continue sa deruleze pentru a vedea rezultatele. In cele din urma, imprima rezultatele finale. Textul este afisat intr-un JTextPane, care face parte dintr-un JScrollPanel care se afla intr-un JPanel independent, pentru a permite derularea

functionalitatii.



* 1. Design decisions

Proiectarea are la baza ideea unei implementari MVC, separand vizualizarile, controlerul si modelele in diferite clase, dar are si parti din alte modele de design. Un exemplu ar fi modelul Bridge care este utilizat pentru a separa selectia serverului bazata pe timp sau pe lungimea cozii pentru clienti.

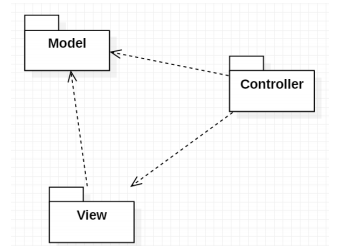
Packages & Relationship Packages

Urmand principiile MVC, proiectul este separat in 3 pachete principale:

● Modelul care are toate clasele necesare modelarii datele necesare pentru indeplinirea sarcinii.

● Controlerul care are toate clasele necesare pentru control, datele pentru a avea rezultatul final.

● Vizualizarea care are toate clasele necesare pentru a afisa informatiile pe ecran si, de asemenea, pentru cititirea intrarii din primul GUI.



Data structures

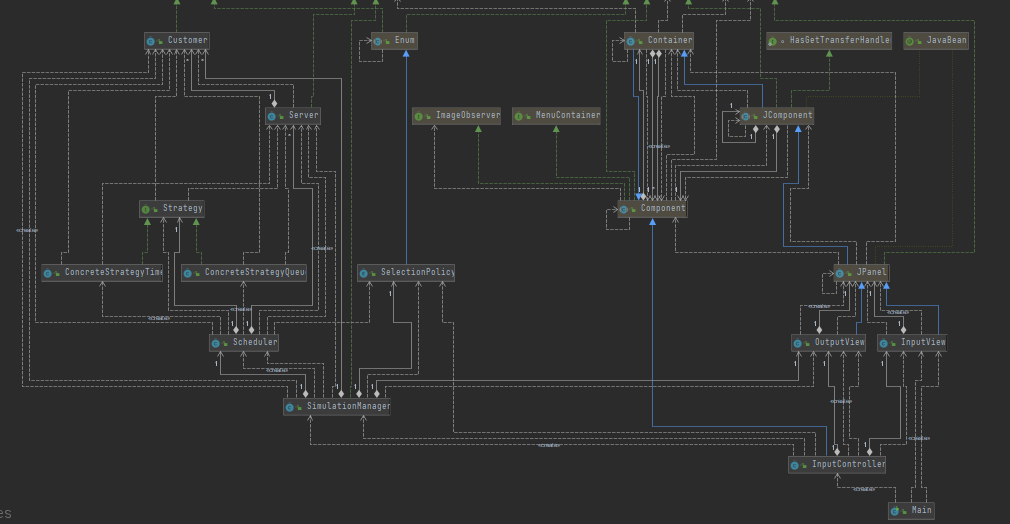
Proiectul a necesitat utilizarea mai multor structuri de date sigure pentru a mentine siguranta firelor intre toate firele, alaturi de multe structuri interesante, dar mai des utilizate. Unele dintre cele mai interesante structurile de date ar include:

● BlockingQueue -> este utilizat pentru a avea cozi sigure pentru fire, de asemenea parametrul volatil este specificat pentru siguranta mai buna.

● AtomicInteger -> utilizat pentru a efectua adaugarea in siguranta intre fire.

UML Diagram

Urmatoarea imagine reprezinta diagrama UML a proiectului:



1. Implementation

Class Design

Principalele clase necesare pentru proiect poate fi impartit in:

● Simulation Manager -> care este utilizat astfel pentru a mentine ordinea intre fire si sa genereze simularea.

● Server -> care se executa asa cum este un fir practic, un model al cozilor.

● SimulationPolicy -> care este o simpla enumerare care permite alegerea intre cele 2 moduri de rulare pentru selectarea sarcinilor.

● Customer -> care modeleaza clientii care se alatura cozilor.

● Scheduler -> care mentine in principal lista serverelor si strategia aleasa.

Interfaces

Singura interfata prezenta in proiect este interfata „Strategy” care este utilizata pentru a face diferenta intre cele 2 moduri de rulare pentru selectia serverului.

Algorithms

Un exemplu de algoritm ar fi cel care alege serverul corect pe baza anumitor strategii. Itereaza prin toate serverele si il alege pe cel care are valoarea maxima aleasa.

In afara de aceasta, nu exista algoritmi de realizare personala si necesari pentru proiecte, dar necesita utilizarea algoritmilor externi precum „Collections.sort ()”.

User interface

Pentru a atinge obiectivul de a avea atat un GUI pentru introducerea valorilor initiale pentru simulare, cat si o interfata pentru afisarea rezultatelor simularii, a trebuit sa fac 2 GUI-uri independente, primul se numeste „InputView” si acesta permite sa introduceti un set corect de valori de pornire. Al doilea se numeste „OutputView” si este apelat doar atunci cand butonul corespunzator strategiei este apasat cu un set corect de valori.

Implementation

Vom lua toate clasele pe rand si le vom analiza toate partile importante:

Scheduler

Rolul "Scheduler-ului" este de a tine evidenta serverelor si a tuturor firelor acestora, mentinand in acelasi timp strategia astfel aleasa.

SimulationManager

Ofera toate functiile necesare pentru a avea loc simularea, alaturi de cateva functii suplimentarenecesare pentru a imprima iesirea atat pe interfata grafica, cat si in fisierul de iesire.

Functia genereaza toti clientii pe baza datelor de intrare date in primul GUI. Dupa generare, de asemenea, clientii sunt sortati astfel incat sa fie in ordinea timpului de sosire, ceea ce ajuta SimulationManager sa selecteze clientii de fiecare data in mod eficient.

Functia ruleaza pe propriul fir si genereaza mai intai timpul mediu de procesare, apoi la fiecare pas, in timp ce ora curenta este mai mica decat limita si exista fie clienti in cozi, fie clienti care asteapta, mai intai proceseaza sarcinile. Apoi calculeaza ora de varf pentru fiecare pas, scrie pentru afisare, scade timpul de procesare pentru prima sarcina din cozi si apoi doarme inca 1s, astfel incat procesul sa se repete.

La final, imprima valorile finale si inchide fisierul.

ConcreteStrategyQueue

Inlocuieste addTask din interfata Strategy pentru a o aplica la cea mai scurta coada gasita.

ConcreteStrategyTime

Inlocuieste addTask-ul din interfata Strategy pentru a-l aplica la cea mai scurt timp de asteptare din cadrul serverelor.

Strategy

Interfata utilizata pentru ConcreteStrategyQueue si ConcreteStrategyTime pentru a modela ambele strategii posibile.

SelectionPolicy

O enumerare utilizata pentru a diferentia care dintre strategii va fi folosita.

Server

Folosit ca un fel de suprapunere peste BlockingQueue cu functionalitate suplimentara care permite gestionarea tuturor serverelor necesare. Functiile principale utilizate sunt addTask () si run () care ofera capacitatea ambelor add task-uri la fir si proceseaza timpul necesar pentru ca fiecare dintre ele sa plece. De asemenea, mentine perioada de asteptare si totalWaitingTime care sunt utilizate pentru tiparirea valorilor finale.

Customer

Folosit pentru a modela clientii intr-un mod care poate fi utilizat in celelalte clase. Are ora sosirii, processingTime si id.

De asemenea, are capacitatea de a fi comparat si tiparit direct ca un sir.

1. Results

Pentru a valida rezultatele simularii, aceasta a fost rulata pe cele 3 exemple furnizate in DOC. Testele sunt urmatoarele:

Test1 (N = 4 Q = 2 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 60 seconds [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 , 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [2, 30] [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 , 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ] = [2, 4])

Successful simulation

Average service time per customer: 3.25

Average waiting time per customer in the server: 3.25

Peak hour: 6

Test2 (N = 50 Q = 5 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 60 seconds [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 , 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [2, 40] [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 , 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ] = [1, 7])

Successful simulation

Average service time per customer: 4.06

Average waiting time per customer in the server: 7.54

Peak hour: 26

Test3 (N = 1000 Q = 20 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 200 seconds [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 , 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [10, 100] [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 , 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ] = [3, 9])

Successful simulation

Average service time per customer: 5.959

Average waiting time per customer in the server: 89.642

Peak hour: 99

1. Conclusions

Consider ca aceast assignment m-a invatat multe despre utilizarea firelor si lucrului simultan, asigurand in acelasi timp siguranta firelor. In consecinta, atribuirea ofera o modalitate de a intelege mai bine modul in care sunt configurate cozile, in modul in care exista 2 strategii utilizate pentru a selecta cozile, fie apeland la coada care pare a fi cea mai scurta, fie evaluand durata exacta si alegand cel mai scurt timp.

In ceea ce priveste evolutiile viitoare, cred ca aplicatia ar putea utiliza o interfata grafica pentru utilizator mult mai performanta, pentru a simula date mai complexe, pentru a analiza in continuare datele si chiar pentru a oferi feedback furnizorului de servicii cu privire la modul de distribuire mai buna a clientilor pentru un timp minim de asteptare.

1. Bibliografie

* [Java Thread Pool Example using Executors and ThreadPoolExecutor (javacodegeeks.com)](https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html)
* <https://stackoverflow.com/>
* [java.util.Timer.schedule() Method - Tutorialspoint](https://www.tutorialspoint.com/java/util/timer_schedule_period.htm)
* <https://www.wikipedia.org/>