DOCUMENTATIE

TEMA *2*

NUME STUDENT: Toderean Razvan Florin

GRUPA: 30222

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 4](#_Toc95297887)

[4. Implementare 5](#_Toc95297888)

[5. Rezultate 9](#_Toc95297889)

[6. Concluzii 10](#_Toc95297890)

[7. Bibliografie 10](#_Toc95297891)

# Obiectivul temei

1. Simulator magazine
2. Utilizarea thread-urilor pentru a putea rezolva acest task.
3. Utilizarea metodelor thread-safe
4. Fiecare casa de marcat trebuie sa reprezinte un thread.

# Analiza problemei, modelare, scenarii, cazuri de utilizare

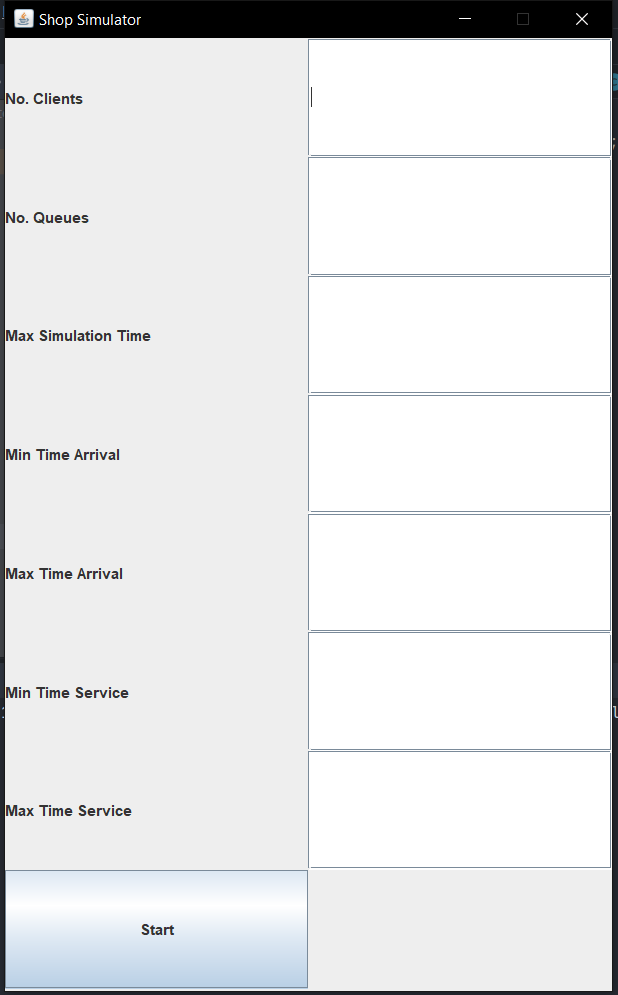
În ceea ce privește implementarea unui program cu threaduri, aceasta poate fi realizată prin extinderea clasei Thread sau prin implementarea interfeței Runnable. În ambele cazuri, trebuie să suprascriem metoda run() care conține codul care va fi executat în thread.

Un exemplu de utilizare a threadurilor în Java ar fi într-un program de procesare a imaginilor. Un thread ar putea fi responsabil pentru citirea imaginii, un alt thread ar putea fi responsabil pentru aplicarea unor filtre asupra imaginii, în timp ce un alt thread ar putea fi responsabil pentru salvarea imaginii procesate. Prin utilizarea threadurilor, aceste sarcini ar putea fi efectuate simultan, ceea ce ar duce la o creștere semnificativă a vitezei de procesare a imaginii.

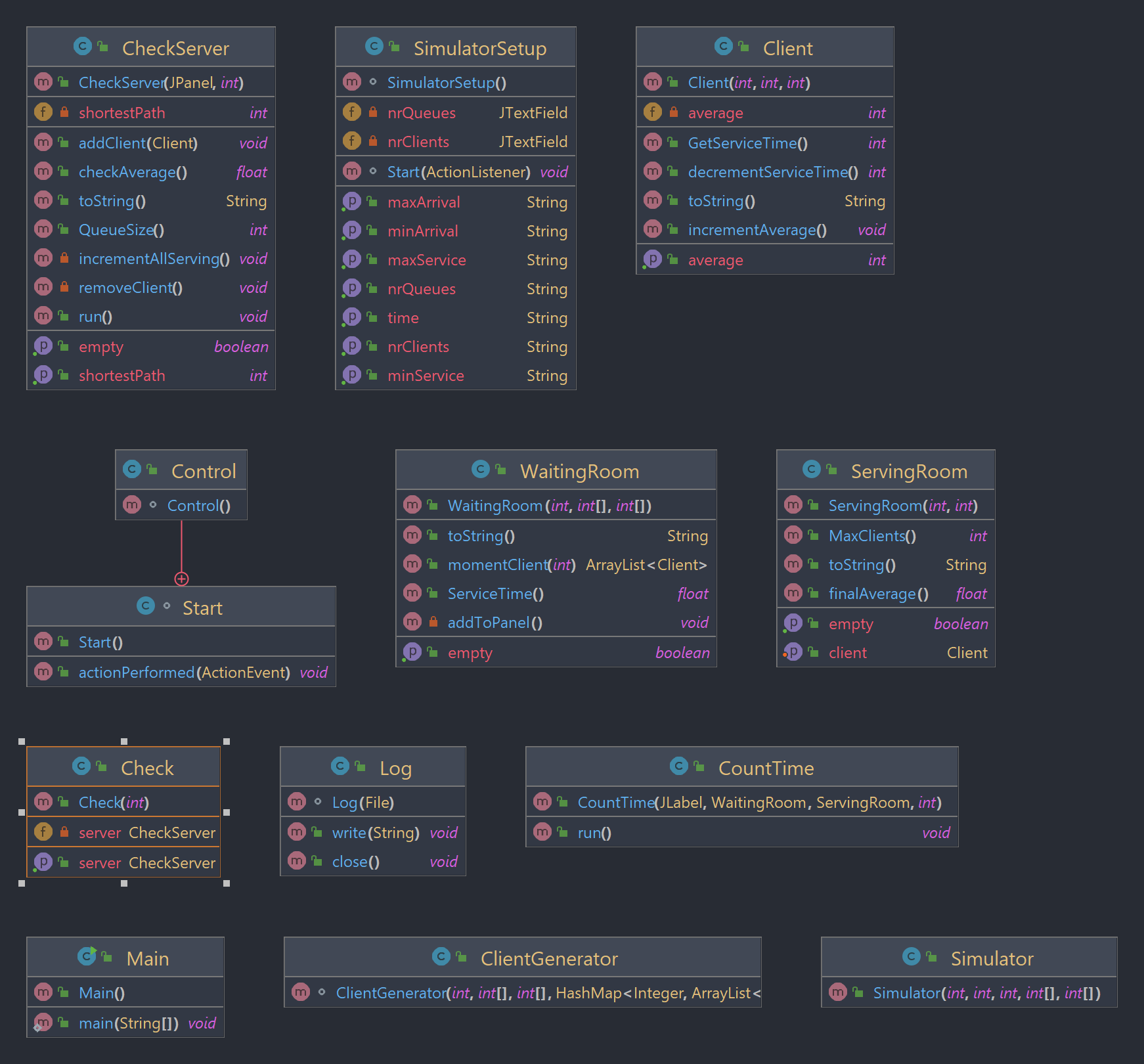
Programul meu simulează cozile de la casa de marcat, fiecare coadă ar putea fi reprezentată de un thread. Astfel, fiecare thread ar fi responsabil pentru procesarea clienților din coadă și ar comunica cu celelalte threaduri pentru a actualiza starea generală a casei de marcat.

Pentru a putea porni simularea trebuie sa cunoastem numarul de client pe care ii vom genera, numarul de case de marcat care se vor afla in magazine, si timpul maxim de simulare(simularea se va opri automat cand timpul maxim de simulare a fost atins, nu conteaza daca nu au fost serviti toti clientii), totodata trebuie sa stabilim un interval in de sosire al clientilor la casa de marcat si un interval de timp in care acestia isi pot achita produsele.

In partea dreapta vom insera datele in ordinea specificata mai sus , butonul de start va face astfel incat sa dispara acest meniu si sa porneasca simularea. Pentru a putea porni alt meniu va trebui sa rulam din nou programul.



# Proiectare



Pentru a creea interfata la acest task am lucrat progresiv in fiecare clasa, in WaitingRoom mi-am creeat un HashMap pentru clientii care sunt in asteptare, cheia este timpul lor de arrive pentru a putea sa le gasesc intr-o complexitate foarte mica, iar valorile acestor chei sunt ArrayList-uri de clienti.

Pentru fiecare casa de marcat am folosit un thread , toate metodele au fost thread-safe astfel incat am folosit variabile de tip AtomicInteger care este o varibila volatila deoarece la o modificare aplicata , rezultatul este vizibil de catre toate thread-urile.

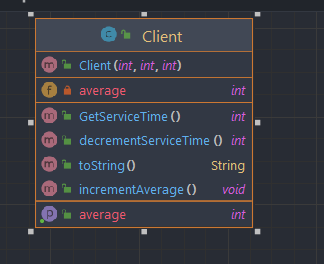
Clasa Check este reprezentarea in GUI al caselor de marcat , clasa cu metodele pentru aceasta casa de marcat este CheckServer.

# Implementare

Clasa client va avea ca attribute un id, arrivalTime si un serviceTime , ca metode implementate avem o metoda care iau serviceTime -ul si alta metoda care preiau acel serviceTime decrementat deoarece l-am facut AtomicInteger, pentru a se decrementa global serviceTime-ul unui client.

O metoda toString dupa modelul pentru logfile

Si metodele pentru average waiting time , care ma ajuta sa incrementez locul fiecarui client la coada casei de marcat.



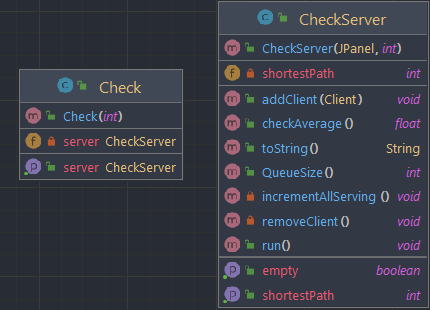
Clasele Check si CheckServer.

CheckServer este clasa pentru a reprezenta o casa de marcat, pentru aceasta clasa mi-am declarat BlockingQueue si o variabila de shorthestPath pentru a tine numarul de clienti la o casa.

In aceasta clasa am implementat metodele addClient si removeClient pentru a adauga clienti la coada dupa criteriul selectat(cea mai scurta coada) si pentru ai sterge in momentul in care acestia si-au terminat serviceTime-ul. Am extins clasa thread, pentru ca fiecare coada sa aiba atrbuit un thread.

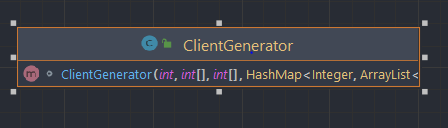
In metoda run() verificam daca se afla sau nu clienti in coada ,iar daca se aflau le decrementam serviceTime-ul. Metodele de incrementAllServings si checkAverage ma ajuta pentru a putea stabili average time waitingul , incrementAllServings imi tine numarul de clienti care se afla la fiecare secunda, iar metoda checkAverage imi returneaza media finala a asteptarii.

QueueSize si isEmpty ma ajuta pentru metoda toString respectiv pentru a ma ajuta sa reprezint timpul care au fost cei mai multi clienti la case.



Clasa ClientGenerator.

In aceasta clasa am doar un id care este AtomicInteger din nou pentru a putea mentine toate masurile de siguranta pentru threaduri. Acest constructor primeste cele doua intervale de timp pentru arrive si pentru service si un HashMap in care voi pune toti clientii generati dupa cheie (adica arriveTime-ul).



Clasele ServingRoom si WaitingRoom. Aceste clase ma ajuta sa imbin clasele anterioare pentru a crea interfata intr-un mod mai interactiv.

In clasa Waiting Room avem metodele momentClient ma ajuta sa sterg acei clientii de la o anumita secunda si sa ii returnez mai tarziu punandu-I in cozi dupa criteriul aplicat. Metoda addToPanel ma ajuta sa ii fac update la fiecare secunda in zona de asteptare.

Metoda ServiceTime imi este utila pentru a face o medie la timpul de service.

Metoda toString am scris-o pentru a creea log file-ul.

In clasa Serving Room am lista cozilor pe care va trebui sa o generez.

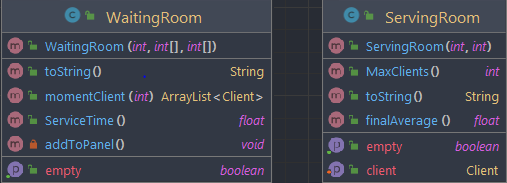
Metoda setClient imi pune clienti in cozi in functie de criteriul decis, mai exact dupa cel mai scurt drum.

Mai exact acesta metoda imi cauta numarul minim de clienti de la o casa ,in momentul in care am gasit minimul el memoreaz indexul acelei case si in acea coada insereaza un client.

finalAverage este metoda care calculeaza media de asteptare , acum de pe toate casele.

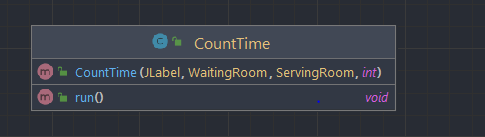
Metoda MaxClients ma ajuta astfel incat imi calculeaza din toate casele cati clienti am in total in fiecare secunda, pentru a putea scoate ora de varf.

La fel toString ma ajuta sa scriu log file-ul.

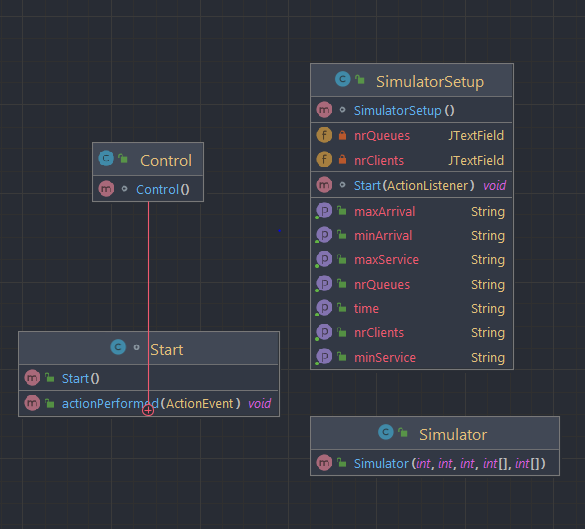


Clasa CountTime imi face ca toate clasele sa lucreze cu un thread la fiecare secunda.

De exemplu in metoda run am declarant un timer care se incrementeaza dupa fiecare secunda de functionare al thread-ului acest lucru face sa mi se actualizeze toata interfata si programul sa ruleze in mod correct. Programul se opreste fie in momentul in care s-a atins timpul limita sau toti clientii au fost serviti. Pentru a creea acel log file am folosit FileWriter.



Clasa SimulatorSetup, Simulator si Control . Sunt clasele care dau start acestui program , in clasa Simulator este creat framework-ul pentru simulare . Simulator Setup este interfata care pregateste simularea introducand datele necesare pentru acest lucru. Iar Clasa Control actioneaza acel ActionListener de la butonul start pentru a da start simularii.



# Rezultate

Test 1

N = 4

Q = 2

𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 60 seconds

[𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ,𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [2, 30]

[𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [2, 4]

Peak Hour is 9

Average service time is 2.25

Average waiting time is 2.25

Test 2

N = 50

Q = 5

𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 60 seconds

[𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ,𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [2, 40]

[𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [1, 7]

Peak Hour is 7

Average service time is 4.04

Average waiting time is 6.78

Test 3

N = 1000

Q = 20

𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 200 seconds

[𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ,𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [10, 100]

[𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [3, 9]

Peak Hour is 100

Average service time is 5.96

Average waiting time is 67.28

# Concluzii

Dupa finalizerea acestei time m-am familiarizat si mai mult cu framework-ul Spring de la Java si totodata am invatat mult mai bine sa folosesc thread-urile intr-un mod mai safe si sa ma asigur ca voi functiona intr-un mod correct si fara a avea erori din cauza unor variabile ne-volatile declarate.

# Bibliografie

<https://dsrl.eu/courses/pt/materials/PT2023_A2_S2.pdf>

<https://stackoverflow.com/questions/541487/implements-runnable-vs-extends-thread-in-java>

<https://stackoverflow.com/questions/36547316/issue-in-implementing-blocking-queue-in-java>