**Documentatie Tema 2**

Ghitun Patricia Roxana

Grupa 30227

**1. Cerințe Funcționale**

Dezvoltarea unei aplicații care afișează în timp real clienți ajungând la cozi. Aceasta va simula un șir de clienți care ajung pentru a primi un serviciu, stau la o coadă, așteaptă un anumit timp pentru a fi serviți, și în final, părăsesc coada. O modalitate pentru a minimiza timpul de așteptare ar fi să se adauge mai multe cozi în sistem (fiecare coadă este considerat să aibă asociat un procesor), dar această abordare crește costul furnizorului de servicii. Se va urmări calcularea timpului de așteptare pentru fiecare client și aflarea momentului în care aceștia părăsesc coada la care au fost distribuiți.

Date de intrare:

• Minimul și maximul de timp de sosire al clienților

• Minimul și maximul de timp de procesare

• Numărul serverelor

• Numărul clienților

• Intervalul de simulare

Date de ieșire:

• Evoluția serverelor pe parcursul simulării (interfață)

• Ora de vârf

• Timpul total al simulării

• Timpul maxim de așteptare

• Timpul mediu de așteptare

**2. Obiective**

Obiectivul acestei teme este proiectarea si implementarea unui sistem de simulare a unei cozi intr-un magazin. Mai exact, aceasta rezolva intr-un mod cat mai eficient problema clientului de a astepta la coada la un magazine . De aceea , clientii care doresc sa se indrepte spre casa de marcat, sunt indreptati spre o casa care are coada cea mai scurta sau spre o casa cu cel mai putin timp de asteptare sau o casa aleasa random .

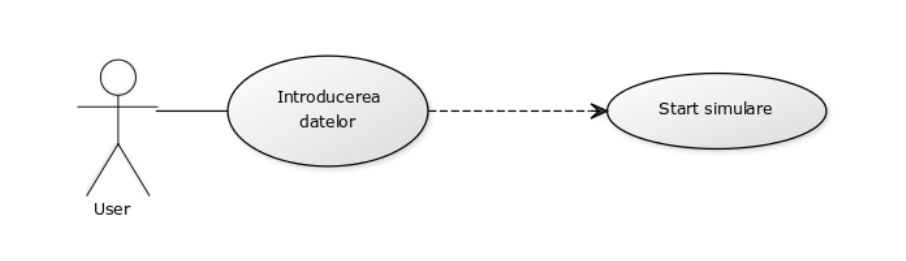
Scopul temei este si aprofundarea principiului de multithreading.

**3. Analiza Problemei**

In primul rand, pentru rezolvarea temei avem nevoie de o buna intelegere a principiului de concurenta in programarea orientata pe obiect. Concurenta este proprietatea de a rula programe in paralel.

Cea mai buna solutie pentru implementarea acestor probleme este folosirea thdread-urilor. Conceptul de thread (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către sistemul de operare. Este folosit în programare pentru a eficientiza execuția programelor, executând porțiuni distincte de cod în paralel în interiorul aceluiași proces. Tehnica prin care un thread asteaptă execuția altor threaduri înainte de a continua propria execuție, se numește sincronizarea threadurilor.

Dragrama use case :



In cazul de fata , utilizatorul introduce datele :

* min Arrival Time : timpul minim la care poate sa ajunga un client la coada
* max Arrival Time : timpul maxim la care poate sa ajunga un client la coada
* min Processing Time : timpul minim in care sta clientul sa fie procesat ( cat timp sta sa plateasca cumparaturile )
* max Processing Time
* nrOfClients : numarul de clienti care vin la magazin
* nrOfQueues : numarul de cozi din magazine
* simulationTime : timpul de simulare al aplicatiei;

Dupa introducerea datelor , se apasa butonul “Start” , aplicatia citeste datele introduse si afiseaza rezultatul algoritmului .

**4. Proiectare**

**4.1 Structuri de date**

O structură de date se referă la metode de organizare a unităților de date în așa fel încât ele să poată fi folosite în mod eficient. Realizarea și menținerea structurilor de date specifice contribuie la îmbunătățirea algoritmilor implementați.

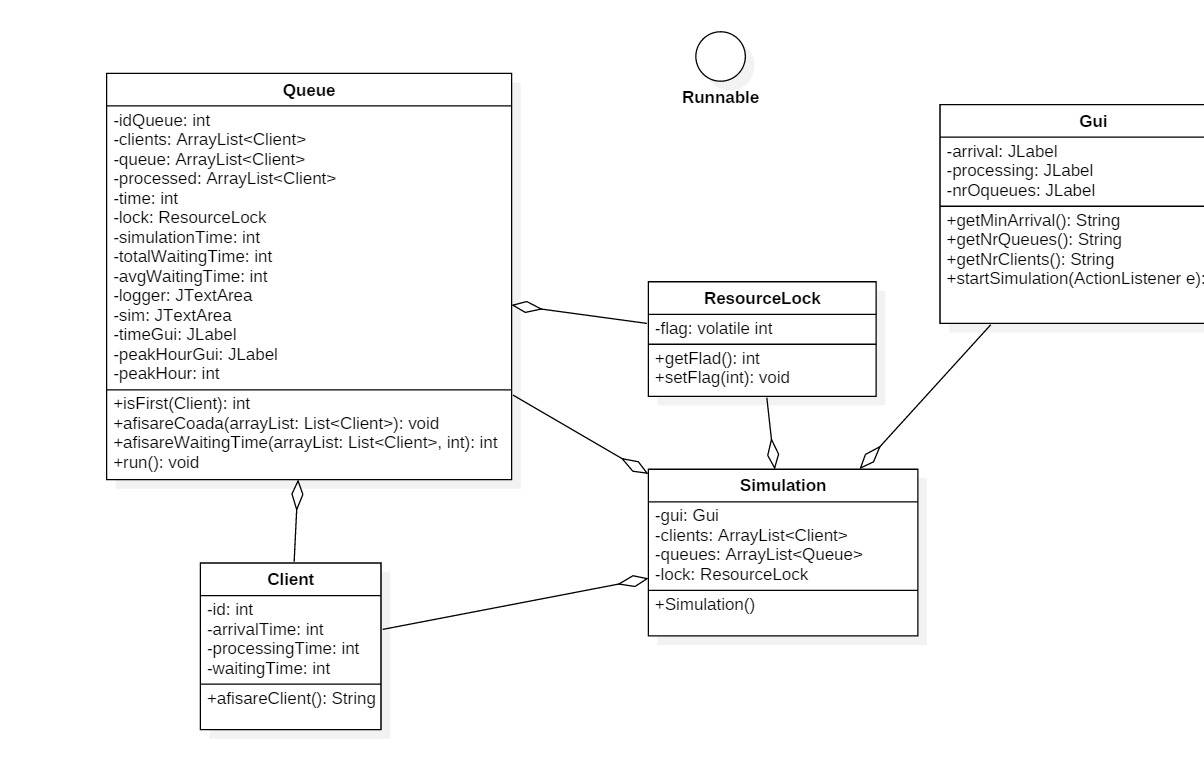
Structurile de date se implementează folosind tipuri de date abstracte , care sunt specificări a unui set de date de un anumit tip, împreună cu operațiile care pot fi executate pe aceste date.

Fiecare Coada din cadrul aplicatiei are un atribut de tip ArrayList<Client> clients ce reprezinta clientii stocati in momentul in care au fost generate , adica clientii ce asteapta sa fie introdusi in coada . Acesti client vor fi adaugati in coada in momentul in care timpul lor de sosire (arrivalTime) este egal cu timpul simularii .

Clientii care au “venit” sunt adaugati intr-o noua lista tot de tip ArrayList<Client> numita queue , coada efectiva cu clientii care au fost generate si adaugati .

La final , in lista tot de tip ArrayList<Client> processed vor fi clientii care au fost procesati , adica cei care au ajuns la casa , au platit cumparaturile si au plecat din coada (clientii care au avut processingTime-ul 0).

**4.2 Diagrama de clasa – Diagrama UML**

****

**5. Implementare**

**5.1 Clasa Client**

Aceasta clasa are patru variabile instanta : id-ul unui client de tip int , arrivalTime sau timpul de sosire de tip int – timpul in care clientul este adaugat in coada in functie de timpul simularii , processingTime sau timpul de servire – cat timp sta clientul la casa pentru a plati alimentele si waitingTime – cat timp a asteptat clientul la coada pentru a fi procesat ( pentru a ajunge primul ) .

In afara metodelor de get si set pentru variabilele instanta care realizeaza accesul la datele clasei din alta clasa , am mai implementat inca o metoda numita afisareClient care afiseaza toate datele despre clientul respectiv . Aceasta metoda va fi folosita ulterior pentru afisarea clientilor pe ecran .

**5.2 Clasa Queue**

Aceasta clasa reprezinta o coada din magazin si are ca atribute principale cele trei liste de client : clients , queue si processed (clientii generate care nu au ajuns inca in coada si care astepta , clientii care au intrat in coada si clientii care au fost procesati si au plecat ).

Pe langa aceste atribute , am mai inclus si cate un id pentru fiecare coada pentru a fi mai usor sa afisez datele pe ecran .

De asemenea , am mai introdus si timpul current , timpul simularii , totalWaitingTime , averageWaitingTime , peakHour , numarul maxim de client la momentul t din coada – max si cateva componente care fac legatura cu interfata grafica a aplicatiei cum ar fi : 2 textArea-uri si 3 label-uri .

Aceasta clasa este clasa de baza a aplicatiei deoarece aici se implementeaza interfata Runnable , ceea ce inseamna ca fiecare instanta a acestei clase va fi executata de catre un thread (fir de executie ) . Clasa trebuie sa implementeze metoda run – metoda din interfata Runnable .

Cand un obiect care implementeaza interfata Runnable , este folosit pentru a crea un nou thread , pornirea lui va determina apelul metodei run in thread-ul respective (thread.start()).

Pentru fiecare obiect din aceasta clasa , se vor cunoaste cele trei liste de client , id-ul ei , si toate datele referitoare la clientii din acea coada .

Metodele importante implementate in clasa Queue :

* public void addNewClient(Client c)

Aceasta metoda va adauga un nou client in lista de client ce asteapta sa intre in coada – in clients . Aceasta metoda este folosita ulterior pentru a adauga clientii generate random in coada de “asteptare”.

* public int isFirst(Client c )

Aceasta metoda verifica daca in lista de clienti care sunt deja in coada mai sunt clienti , iar daca da verifica daca primul client din coada este acelasi cu cel dat ca parametru . Daca se indeplineste aceasta conditie atunci metoda va returna valoarea 0 , altfel valoarea 1 .

* public void Run()

Aceasta metoda este cea implementata din interfata Runnable . In metoda asta , se intampla toata “actiunea” proiectului . Cat timp , timpul current este mai mic decat timpul de simulare dat din interfata grafica de catre utilizator , se intampla urmatoarele :

* pentru fiecare client din lista de client generati care nu au intrat inca in coada , se verifica daca arrivalTime-ul lor este egal cu timpul current al simularii , iar daca da atunci clientul respective este adaugat in coada efectiva – queue .
* Dupa ce se adauga toti clientii care sunt in asteptare , in coada , se parcurge lista de client aflati deja in coada cu un iterator si se verifica daca clientul curent este primul element din lista . Daca se intampla acest lucru , atunci inseamna ca clientul este chiar la casa de marcat si plateste cumparaturile , ceea ce inseamna ca ii scade processingTime-ul

Cand clientul de la casa ajunge sa aiba un processingTime egal cu zero , atunci acesta pleaca din coada – este sters din queue si din clients si este adauagat in processed .

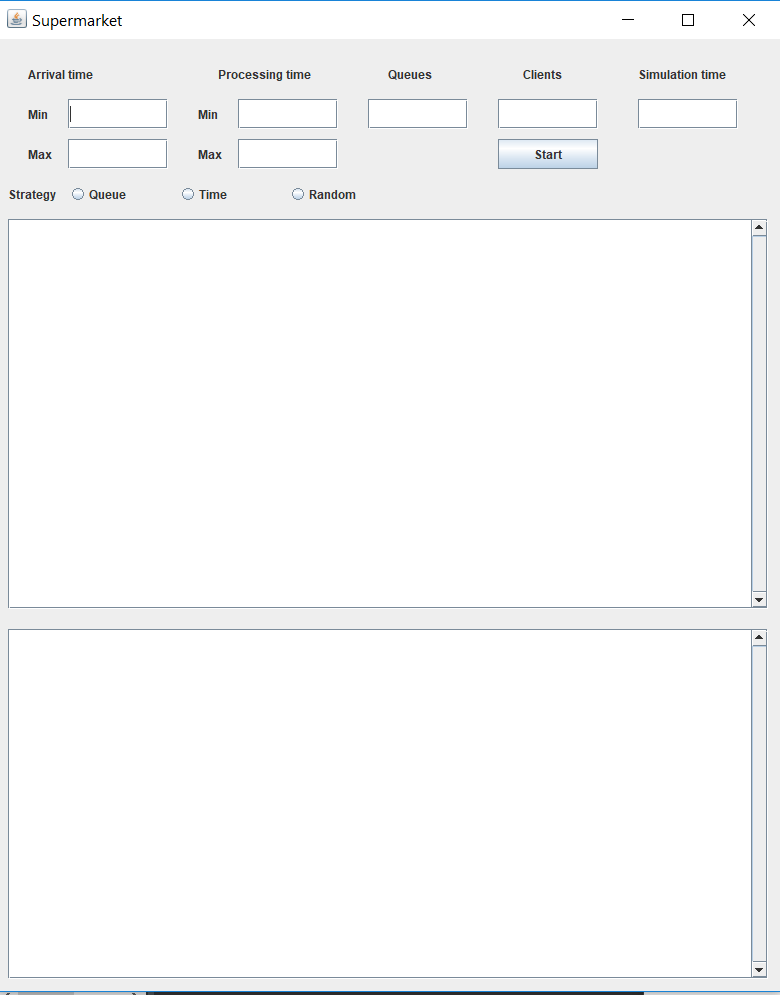
Daca clientul curent nu este primul client din coada ( nu se afla la casa de marcat ) atunci ii creste waitingTime-ul .

La final , thread-ul curent doarme pentru o secunda , ceea ce inseamna ca aceste lucruri sunt verificate pentru fiecare secunda .

**5.3 Clasa Gui**

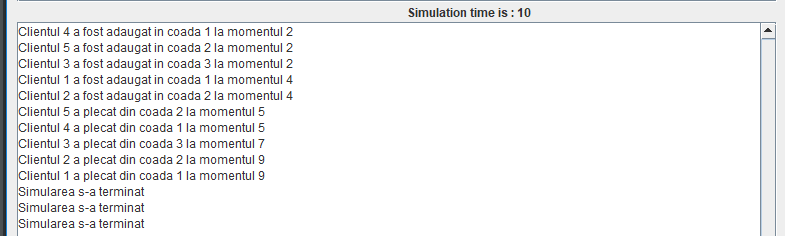
Aceasta clasa implementeaza interfata grafica pentru utilizator si defineste mai multe metode care vor transmite datele in clasa Simulation pentru a face legatura intre simulare si interfata .

Metodele respective sunt metodele de get ale datelor introduse de catre utilizator precum : getMinArrival, getMaxArrival , getMinProcessing, getMaxProcessing ,getNrQueues , getSimulationTime , getNrClients , getTime(pentru a transmite timpul simularii) ,getLogger , getTextAreaSimulation , getRadioQueue , getRadioTime , getRadioRandm(aceste trei metode returneaza starea radioButton-urilor – daca au fost selectate sau nu ) .

La pornirea aplicatiei va aparea urmatoarea fereastra : 

In partea de sus se vor introduce datele de catre utilizator dupa care se va apasa start .

In primul textArea vor fi afisate cozile in timp real cu clientii lor , iar in cel de al doilea se va afisa Loger-ul de date al aplicatiei .

De exemplu , pentru o generare random de 5 clienti la 3 cozi cu un timp de simulare 10 , logger-ul va afisa urmatoarele : 

**5.4 Clasa Simulation**

In aceasta clasa se face legatura dintre interfata grafica si algoritmul cozilor . Ca atribute avem o instanta de clasa Gui si doua ArrayList<Client>-uri de client . Una dintre este este folosita pentru a stoca clientii generati random in momentul in care se apasa butonul “Start” inainte de a fi adaugati in cozi , iar cealalta este petnru a stoca cozile in functie de numarul introdus de cozi .

In constructorul clasei Simulation se implementeaza metoda startSimulation din Gui si se preiau datele introduse de catre utilizator ce urmeaza sa fie folosite pentru a genera o lista de client . Pe langa aceste date , tot aici in constructor in functie de radioButton-ul ales pentru strategie ( Queue, Time sau Random ) se va seta o variabila declarata global (strategy) 0 – random , 1 – queue sau 2 – time.

Generarea clientilor random :

Se foloseste o instanta de Clasa Random denumita random = new Random() dupa pentru fiecare client nou generat I se va calcula arrivalTime-ul in ca random.nextInt(maxArrivalTime-minArrivalTime)+minArrivalTime , si similar pentru processingTime-ul sau .

Dupa ce au fost generate random acele valori , se creeaza un nou Client care are in constructor identificatorul sau si cele doua valori generate random . Dupa crearea clientului , acesta va fi adaugat in lista de client generati (clients) .

Dup ace avem lista cu clientii generati , se vor crea cozile in functie de numarul de cozi date de catre utilizator . Aceste cozi vor fi umplute cu client in functie de strategia data din interfata (strategy) .

Daca strategia este 0 , random , atunci fiecare client din lista clients va fi adaugat in lista de cozi pe o pozitie aleasa random . Se va folosi metoda implementata in clasa Queue addNewClient pentru a adauga clientul in coada .

Daca strategia este 1 , atunci clientul va fi adaugat la cea mai scurta coada . Pentru fiecare client , pentru inceput se initializeaza o variabila minSize cu dimensiunea primei cozi dupa care se parcurge lista de cozi si daca se gaseste o coada cu size-ul mai mic , atunci acesta se schimba . In final clientul va fi adaugat pe pozitia cozii cu cei mai putini clienti .

Daca strategia este 2 , atunci clientul nou va fi mereu adaugat la coada cu cel mai putin timp de asteptare , iar pentru asta se verifica daca timpul de asteptare al unei cozi este mai mic decat cel ales prima data . In clasa Queue a fost implementata o metoda getQueueWaitingTime care returneaza timpul de asteptare pentru fiecare coada .

**5.5 Clasa ResourceLock**

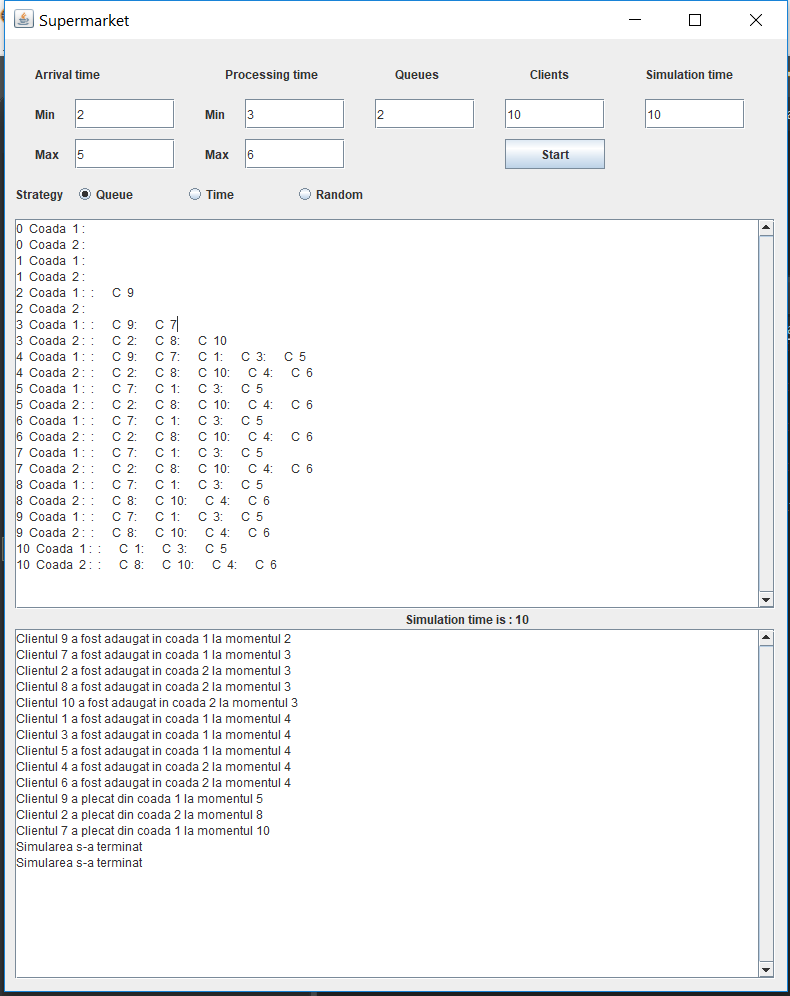
Aceasta clasa a fost folosita pentru a crea un lock pentru threaduri ca sa nu se execute afisarile deodata. Astfel , se va folosi un flag care va reprezenta id-ul cozii , iar daca in metoda run , id-ul cozii nu este egal cu flag-ul , atunci thread-ul respectiv va astepta .

**5.6 Clasa Main**

Aceasta este ultima clasa a aplicatiei si contine doar o instanta de clasa Simulation .

**6. Rezultate**

Rezultatele acestei aplicatii pot fi observate ruland aplicatia , astfel utilizatorul poate sa observe evolutia cozilor in timp real , dar de asemenea , in consola sunt afisate mai in detaliu evenimentele petrecute in momentul rularii programului .



**7. Concluzii**

Pentru o imbunatatire a aplicatiei , se poate lucra mai mult la interfata cu utilizatorul pentru a devein mai “user-friendly” . De exemplu , clientii pot avea forma unor buline iar in dreptul lor sa fie timpul de servire si sa fie vizibil cum acesta scade pe masura ce se scurge timpul .

**8. Bibliografie**

<https://stackoverflow.com>

<https://docs.oracle.com/javase/tutorial/uiswing/>

<https://en.wikipedia.org/wiki/Swing_%28Java%29>/

<https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html>

<https://www.tutorialspoint.com/java/java_multithreading.htm>

<https://www.javaworld.com/article/2077138/java-concurrency/introduction-to-java-threads.html>

<http://javabypatel.blogspot.ro/>