

**Simulator de cozi**

**Documentație**

Pop Crina-Maria

Grupa 30228, An 2

**Cuprins**

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie

**1. Obiectivul temei**

Obiectivul principal al celei de-a doua teme de laborator a fost să gândim, să proiectăm și să implementăm un simulator de cozi care să arate funcționarea bazată pe aceste structuri de date a unor cozi din lumea reală, cu scopul de a minimiza timpul de așteptare al clienților. Simulatorul primește ca date de intrare de la utilizator numărul de clienți care așteaptă, numărul de cozi disponibile, timpul de simulare, timpul minim și maxim de sosire al unui client, cât și timpul minim și maxim de servire sau de procesare al unui client (mai exact, timpul cât un clien stă la ”casă” și este servit). Plecând de la aceste date de intrare, aplicația noastră genereaza clienți cu timp aleator de sosire și de procesare (un număr din intervalul dat), ca mai apoi să fie pus să aștepte până când timpul lui de sosire este egal cu timpul actual al simulării., moment în care este trimis către casa cu timp minim de așteptare. Până un client este trimis la o coadă să fie ”servit”, toate cozile sunt închise. Generarea rezultatelor se face fie în interfața grafică în timp real, fie într-un fișier text unde se vor găsi evenimentele de la fiecare secundă din simulare, ceea ce s-a întâmplat cu cei care așteptau. Una dintre cerințele principale ale acestei teme a fost folosirea thread-urilor pentru fiecare coadă în parte.

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

Modelarea problemei am facut-o pornind de la scheletul de cod pus la dispoziție în cadrul laboratorului. Cozile sunt proiectate pentru a putea primi clienți pe care să îi proceseze. Pe lângă acestea, mai avem un Scheduler care se ocupă de asignarea clienților la câte o coadă în funcție de timpul de sosire al acestuia și de timpul actual al simulării. Tot scheduler-ul estte cel care închide sau deschide cozile. Pentru a genera clienții de la ocoadă am proiectat un Simulation manager care se folosește de scheduler. Citirea datelor de intrare se face din interfața grafică. Un exemplu de date de intrare care se poate folosi pentru a testa aplicația este următorul:

N = 4;

Q = 2;

Tsim = 60;

Tarivmin = 2;

Tarivmax = 30;

Tservmin = 2;

Tservmax = 4;

Un caz de utilizare al simulatorului nostru este prezentat mai jos:

Use-Case: simularea pentru un set de date

Actor principal: utilizatorul

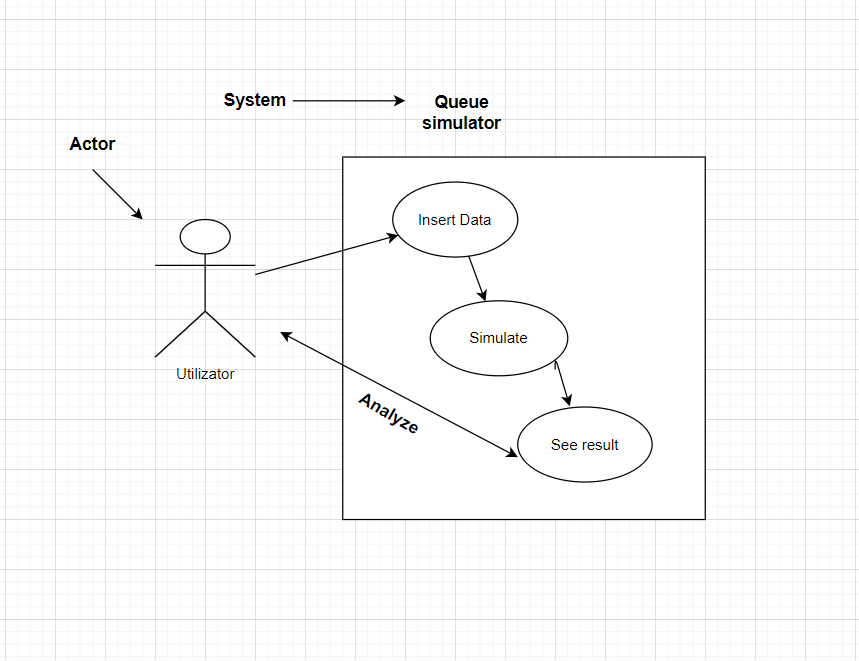
Scenariul principal corect de funcționare:

1. Utilizatorul introduce datele necesare în interfață
2. Utilizatorul selectează butonul ”start”
3. Simulatorul afișează în fiecare secundă starea cozilor deschise sau închise, cât și clienții care așteaptă.

Scenariul alternativ de funcționare: date de intrare incorecte

Utilizatorul introduce alte date de intrare după apăsarea butonului ”clear”

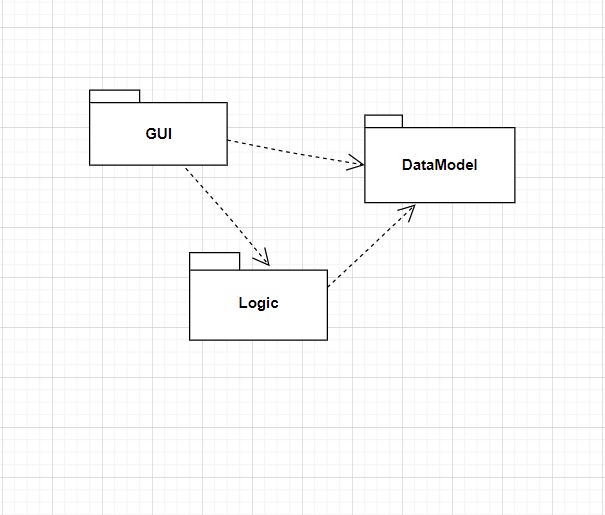
Scenariul se intoarce la pasul 1.



**3. Proiectare**

Proiectarea claselor necesare nu a fost atât de dificilă, având în vedere faptul că am pornit de la câteva idei care m-au ajutat să îmi dau seama de ce este nevoie pentru aplicația dorită. Pe lângă clasele propuse, am mai adăugat câteva clase care am considerat că îmi sunt necesare în structurarea aplicației.

Până să ajung la clasele propriu-zise mi-am creat pachetele necesare pentru implementarea unui design-pattern cunoscut. Am mers tot pe logica de Model-View-Controller, întrucât mi-am creat 3 pachete și anume: Logic (în care are loc simularea propriu-zisă), DataModel (unde am pus structurile de date de care am avut nevoie) și GUI (pentru interfața grafică).



După crearea pachetelor, am făcut împărțirea pe clase.

În pachetul DataModel am inclus clasele Client și Coadă.

Clasa Client modelează un client din lumea reală care are un id unic (ca un nume propriu), timpul se sosire la coadă ”t\_arrival” și timpul necesar servirii unui client, ”t\_service” (pentru scanarea produselor și achitarea sumei de bani).

Clasa Coadă are strcutura unei cozi reale, iar pentru aceasta au fost definite ca atribute o structura de date de tip BlockingQueue<Client> în care se adaugă clienții care vin la coada respectivă si waitingTime care are rolul de ține minte timpul de așteptare de la coada respectivă (adică timpul de procesare a clientului asignat cozii, care scade), pentru ca mai apoi să determinăm coada cu timp minim de așteptare. S-a folosit structura BlockingQueue pentru ca folosește colecții sincronizate, de asemenea pentru waitingTime s-a folosit AtomicInteger, întrucât este thread safe și să nu avem probleme atunci când două thread-uri încearcă să acceseze simultan aceste câmpuri, pentru a nu avea date inconsistente.

În pachetul Logic am inclus clasele Scheduler, SimulationManager și Sort.

Clasa Scheduler este responsabilă de crearea cozilor și a thread-urilor pentru fiecare coadă și de asignare a clienților la o coadă în funcție de strategia aleasă. Ca atribute au fost definite o lista de clienti care va reține un număr maxim de clienți per coadă, un număr maxim de cozi care pot fi create și un obiect de tip Strategy care poate alege în funcție de timpul minim de așteptare de la o coadă sau de coada cu număr minim de clienți.

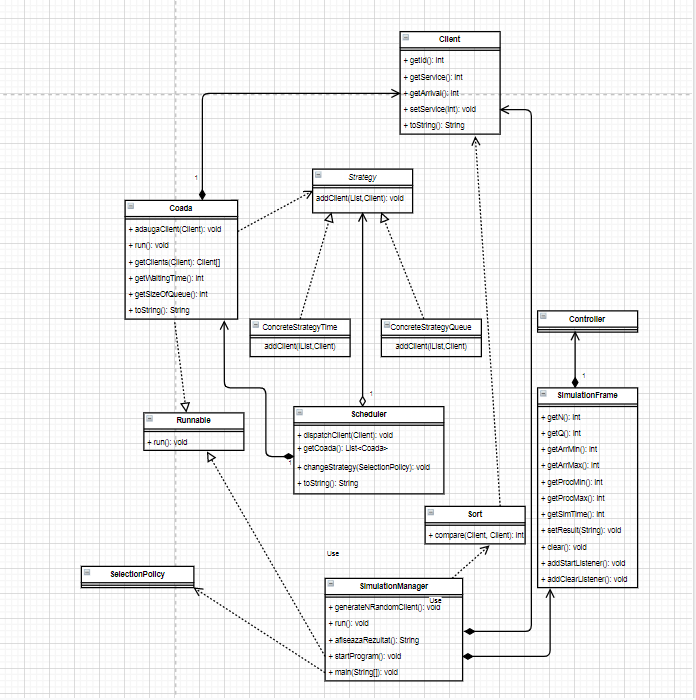
Clasa SimulationManager este simulatorul propriu zis al aplicației noastre, care ne creează clienții dându-le valori random timpilor de sosire si servire, pentru ca mai apoi să îi trimită la o coadă. Aici am definit mai multe atribute și anuume: ”timeLimit” care este timpul total de simulare, ”numberOfQueues” pentru numărul cozilor, ”numberOfClients” pentru numărul de clienți, ”minProcessingTime și ”maxProcessingTime” pentru intervalul de timp înn care un cliient poate fi servit și ”minArrivingTime și ”maxArrivingTime” pentru intervalul de timp în care un client poate sosi. Pe lângă acestea, am mai avut nevoie de un atribut pentru scrierea în fișier, „myWriter”, un obiect de tip Scheduler, o listă în care ținem clienții din coada de așteptare care au fost generați de generateNRandomClient() și un ”frame” pentru interfața grafică.

Clasa Sort am implementat-o pentru a sorta clienții din coada de așteptare ca să fie trimiși să fie procesați în fncție de timpul de sosire, mai exact crescător.

Pachetul GUI conține clasele SimulationFrame, Controller, ConcreteStrategyTime, ConcreteStrategyQueue și interfața Strategy.

În SimulationFrame mi-am definit toate componentele interfeței mele grafice și câteva metode pentru a obține datele de intrare, iar clasa Controller este responsabilă cu implementarea ascultătorilor pentru butoane. Interfața Strategy definește o singura mtodă abstractă, pe care clasele ConcreteStrategyTime și ConcreteStrategyQueue o implementează concret în funcție de ce SelectionPolicy-ului necesar (cel pentru coada cu timp minim de așteptare, sau pentru coada cu clienți minim)

**Diagrama UML**

****

**4. Implementare**

**4.1 Metode**

1. **Clasa Client**

* getId() ne returnează câmpul id.
* getArrival() returnează câmpul t\_arrival
* getService() returnează t\_service

1. **Clasa Coada**

* adaugaClient() ne adaugă clientul în colecția de tip BlockingQueue și adună la waitingTime timpul de servire al clientului adăugat
* run() aceasta metodă este implementată aici, întrucât clasa noastră implementează interfața Runnable ca să poată obiectele de tipul thread să lucreze cu clienții din coada asociată lor.

public void run() {  
 while(true) {  
 while (!clienti.isEmpty()) {  
 try {  
 Thread.*sleep*(1000);   
 waitingTime.getAndDecrement();   
 int timp = clienti.peek().getService();  
 timp--;  
 clienti.peek().setService(timp);  
 if(clienti.peek().getService() == 0) {  
 clienti.take();  
 }  
 } catch (Exception e) {}  
 }  
 }  
}

* getWaitingTime() returnează valoarea câmpului waitingTime
* getSizeOfQueue() am implementat-o pentru a o folosi în implementarea metodei addClieti() din clasa ConcreteStrategyQueue ca să pot afla numărul minim de clienți toate cozile
* toString() este folosită la afișarea rezultatului. Pentru a știi daca o coadă este închisă sau nu am verificat daca lista de clienti e goala și timpul de așteptare să fie 0.

1. **Clasa Scheduler**

* changeStrategy() este folosită pentru a verifica tipul strategiei aplicate în alegerea cozii la care să fie pus clientul. În funcție de parametrul pe care il are se va apela una dintre metodele ConcreteStrategyTime sau ConcreteStrategyQueue
* dispatchTask() este folosita doar pentru a asigna clienții la cozi (apeleaza metoda addClient())
* getCozi() returneaza lista de cozi din această clasă
* toString() contribuie la crearea stringului pentru afisare

public String toString() {  
 String rez = "";  
 for(int i = 0; i < maxNrCozi; i++) {  
 rez = rez + "Queue " + i + ": " + cozi.get(i).toString() + "\r\n";  
 }  
 return rez;  
}

1. **Clasa SimulationManager**

* generateNRandomClients() ne generează clienții aleator și ii pune în ArrayList-ul care ține minte acești clienți
* run() este suprscrisă din nou această metodă, întrucât clasa SimulationManger implementeaza interfața Runnable ca să poată să se creeze un thread care să țină timpul de simulare și să se ocupe de trimiterea clienților la coada cu timp minim de așteptare. Tot aici avem și partea de scriere în fișier/interfață.

public void run() {  
 int currentTime = 0;  
 int servTime = 0;  
 int size = generatedClients.size();  
 Iterator<Client> it = generatedClients.iterator();  
 Client c = it.next();  
 while(currentTime < timeLimit) {  
 while(c.getArrival() == currentTime) {  
 scheduler.dispatchClient(c);  
 servTime += c.getService();  
 it.remove();  
 if(it.hasNext()) {  
 c = it.next();;  
 } else {  
 break;  
 }  
 }  
 try {  
 myWriter.write(afiseazaRezultat(currentTime));  
 *frame*.setResult(afiseazaRezultat(currentTime));  
 } catch(Exception e){}  
 //System.out.println(afiseazaRezultat(currentTime));  
 currentTime++;  
 try {  
 Thread.*sleep*(1000);  
 } catch(Exception e) {}  
  
 if(c.getService() == 0 && scheduler.isEmpty()) {  
 break;  
 }  
 }

* toString() din această clasă se va ocupa de afișarea formatului final pentru rezultatele obținute
* startProgram() este o metodă pe care am creat-o ca să poată fi apelată la apăsarea butonului start din interfață
* main() în main ne creăm doar fereastra pentru interfața noastră și o facem vizibilă.

1. **Clasele ConcreteStrategyTime și ConcreteStrategyQueue**

* Aceste doua clase definesc comportamentul specific din metoda addClient care se află în interfața Strategy. Varianta pentru ConcreteStrategyTime este următoarea:

public void addClient(List<Coada> cozi, Client c) {  
 int poz = 0;  
 int min = cozi.get(0).getWaitingTime();  
 for(int i = 1; i < cozi.size(); i++) {  
 if(min > cozi.get(i).getWaitingTime()) {  
 min = cozi.get(i).getWaitingTime();  
 poz = i; //salvam pozitia cozii cu cel mai mic timp de asteptare  
 }  
 }  
 cozi.get(poz).adaugaClient(c);  
}

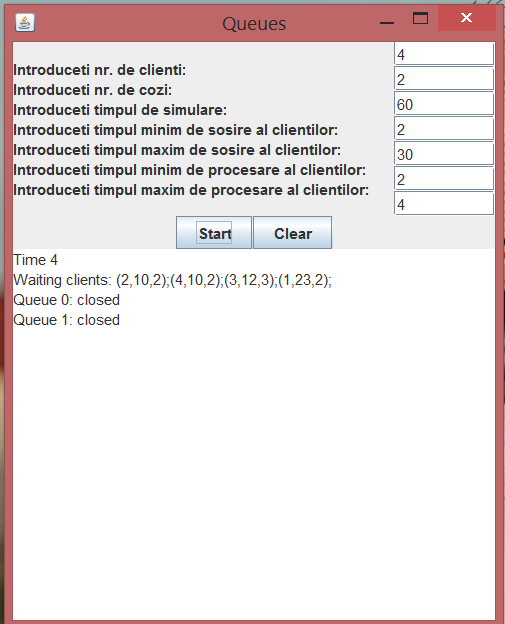
**4.2. GUI**

Interfața grafică pe care am realizat-o este una simplă, în care se pot introduce datele de către utilizator, ca mai apoi să se apese butonul de ”start” și să pornească simularea. Vom observa cum pe rând apar rezultatele în zona de text și se tot actualizează pe măsură ce trecet timpul de simulare. Butonul ”clear” este folosit pentru a șterge ceea ce avem deja introdus în campurile de date, pentru a se putea porni o nouă simulare.

Pentru ascultători am optat să creez clase interne din care să apelez metoda de care am nevoie să se execute atunci cînd este apăsat unul dintre butoane. Pentru ”start” avem următoarea clasă:

class StartListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 SimulationManager.*startProgram*();  
 }  
}

Aici avem o imagine cu interfața grafică pe timpul unei simulări.



Pentru a-mi construi interfața cu utilizatorul am pornit de la câteva idei pe care le-am folosit în semestrul I, schimbând doar unele lucruri. Nu este o interfață foarte complicată, stilizată, pentru că am mers de la început pe ideea de funcționare corectă, lăsând la o parte design-ul. Prin acest fapt, eu consider că este o interfață personalizată, care ar mai putea suferi totuși îmbunătățiri. M-am folosit de componentele din pachetul javax.swing si java.awt pentru a o crea.

Astfel interfața mea conține următoarele elemente:

**Frame-ul**: rama în care se adaugă toate componentele din View, este baza interfeței grafice. Acest frame l-am instanțiat în main-ul aplicației mele, iar tot acolo am denumit această fereastră și am setat opțiunea ca la apăsarea butonului ”X” ea să se închidă.

**Panel-ul**: aici am folosit mai multe componente panel, întrucât am utilizat gestionarul de aranjare BoxLayout, și pe fiecare dintre panel-uri am aduagat câteva componente.

**Butoane:** interfața mea conține 7 butoane, cu diferite funcționalități:

* Start – folosit pornirea simulării
* Clear – sterge tot ce este in zonele de text

**TextField și TextArea:** se poate observa că există 7 zone de text, utilizate pentru a prelua datele de la utilizatr și un textArea pentru a afișa rezultatele simulării.

**Labels:** etichetele le-am folosit pentru a ajuta utilizatorul să știe cum să utilizeze simulatorul de cozi.

**5. Rezultate**

Pentru a testa funcționarea aplicației proiectate am utilizat setul de date de intrare din tabelul aflat în descrierea temei. Am reușit să generez câte un fișier pentru fiecare pas, aplicația funcționează chiar și pentru date mari de intrare.

Fișierul1:

N = 4;

Q = 2;

Tsim = 60;

Tarivmin = 2;

Tarivmax = 30;

Tservmin = 2;

Tservmax = 4;

Ieșire: (o parte din ea)

Time 0

Waiting clients: (3,16,2)(1,23,3)(2,26,3)(4,29,2)

Queue 0: closed

Queue 1: closed

Time 1

Waiting clients: (3,16,2)(1,23,3)(2,26,3)(4,29,2)

Queue 0: closed

Queue 1: closed

Time 2

Waiting clients: (3,16,2)(1,23,3)(2,26,3)(4,29,2)

Queue 0: closed

Queue 1: closed

Time 3

Waiting clients: (3,16,2)(1,23,3)(2,26,3)(4,29,2)

Queue 0: closed

Queue 1: closed

Time 4

Waiting clients: (3,16,2)(1,23,3)(2,26,3)(4,29,2)

Queue 0: closed

Queue 1: closed

Time 5

Waiting clients: (3,16,2)(1,23,3)(2,26,3)(4,29,2)

Queue 0: closed

Queue 1: closed

.....

Time 15

Waiting clients: (3,16,2)(1,23,3)(2,26,3)(4,29,2)

Queue 0: closed

Queue 1: closed

Time 16

Waiting clients: (1,23,3)(2,26,3)(4,29,2)

Queue 0: [(3,16,2)]

Queue 1: closed

Time 17

Waiting clients: (1,23,3)(2,26,3)(4,29,2)

Queue 0: [(3,16,1)]

Queue 1: closed

Time 18

Waiting clients: (1,23,3)(2,26,3)(4,29,2)

Queue 0: closed

Queue 1: closed

....

Time 23

Waiting clients: (2,26,3)(4,29,2)

Queue 0: [(1,23,3)]

Queue 1: closed

Time 24

Waiting clients: (2,26,3)(4,29,2)

Queue 0: [(1,23,2)]

Queue 1: closed

Time 25

Waiting clients: (2,26,3)(4,29,2)

Queue 0: [(1,23,1)]

Queue 1: closed

Time 26

Waiting clients: (4,29,2)

Queue 0: [(2,26,3)]

Queue 1: closed

Time 27

Waiting clients: (4,29,2)

Queue 0: [(2,26,2)]

Queue 1: closed

Time 28

Waiting clients: (4,29,2)

Queue 0: [(2,26,1)]

Queue 1: closed

Time 29

Waiting clients:

Queue 0: [(4,29,2)]

Queue 1: closed

Time 30

Waiting clients:

Queue 0: [(4,29,1)]

Queue 1: closed

Average service time: 2.5

După cum se poate observa, atunci când nu mai avem clienți care așteaptă simularea se termină. Nu am reușit să o fac să continue tot timpul până când nici cozile nu mai au clienți, așa că am lasat rezultatul așa (uneori functionează și se incheie doar când toate cozile sunt inchise, alteori rămâne ultimul client cu timp de servire 1). Average time-ul nu este unul bun, însă nu am găsit o metodă mai bună pentru a-l calcula.

**6. Concluzii**

Din punctul meu de vedere, aceasta a fost o temă mai complicată pentru mine, întrucât nu am mai lucrat cu Thread-uri până acum și mi-a fost greu la început să înțeleg ce aș avea de implementat. Realizarea proiectului a necesitat câteva căutări suplimentare, întrucât m-am lovit de anumite excepții pe care a trebuit să le rezolv. Ca o îmbunătățire, s-ar putea modifica partea de calcul a timpul mediu de așteptare și rezolvarea problemei finalizării simulării doar după ce nu mai există clienți în nici o coadă.

**7. Bibliografie**

[**https://docs.oracle.com/javase/7/docs/api/**](https://docs.oracle.com/javase/7/docs/api/)

[**https://www.baeldung.com/java-generating-random-numbers-in-range**](https://www.baeldung.com/java-generating-random-numbers-in-range)

[**https://www.w3schools.com/java/java\_files\_create.asp**](https://www.w3schools.com/java/java_files_create.asp)

[**https://stackoverflow.com/questions/11888496/file-writer-doesnt-work**](https://stackoverflow.com/questions/11888496/file-writer-doesnt-work)

[**https://www.w3schools.com/java/java\_threads.asp**](https://www.w3schools.com/java/java_threads.asp)