DOCUMENTATIE
TEMA 2
SIMULARE COZI

Nume: Stefan
Prenume: Florina Diana
Grupa: 30227

Cuprins

1. Cerinte functionale

Descrierea si implementarea unei aplicatii de simulare si analizare a unui sistem de tip coada creat cu scopul de determinare si minimizare a timpului de astepare a clientilor, avand in vedere mai multe aspecte care ar putea sta la baza unei mai bune fluidizari a clientilor.

Aceasta aplicatie trebuie sa includa o interfata grafica care permite vizualizarea evolutiei in timp real a cozilor respective, de asemenea datele de intrare (precum intervalul in care un client soseste la casa, intervalul in care un client este servit, numarul de clienti, numarul de cozi, timpul de procesare , dar si viteza de procesare) vor fi introduse tot prin intermediul interfetei, cum de asemenea vor fi afisate si datele de iesire (precum media timpului de asteptare si ora de varf) .

2. Obiective

2.1 Obiectiv principal

Tema curentă are ca obiectiv principal proiectarea unui sistem bazat pe multiple cozi de așteptare, sistemul având un comportament specific situațiilor din lumea reala. Un sistem de tip coada este adesea folosit pentru a modela diferite scenarii din realitate. Coada este o structură de date abstractă, pentru care operatia de inserare a unui element se realizează la un capăt, în timp ce operatia de extragere a unui element se realizează la celălalt capăt.

Aplicatia trebuie sa simuleze o serie de clienti sosind cu o anumita cerinta, care se pun la coada, sunt serviti si la final parasesc coada. Aceasta aplicatie trebuie sa afiseze timpul mediu de asteptare a unui client la o coada.Pentru a calcula timpul de asteptare, vom avea nevoie de un timp de sosire, de terminare si de prelucrare. Timpul de sosire si timpul de prelucrare depind individual de clienti. Timpul de iesire, terminare depinde de numarul de cozi, de numarul de clienti asezati la coada si cerintele pe care ei le au.

Interfata grafica a aplicatiei permite utilizatorului sa introduca minimul si maximul timpului de sosire a clientilor, dar si minimul si maximul timpului de servire a clientilor, numarul de case disponibile. Dupa generarea fiecarui client cu timpurile lui de sosire si servire alocate random, acesta merge la casieria aleasa prin intermediul unei strategii aleasa tot de catre utilizator: casa cu timpul de asteptare cel mai scurt, casa cu cei mai putini clienti sau random. Acesta sta la coada si astepta sa fie servit. In partea de jos a aplicatiei se pot vedea detaliile acestor procese, intrarea unui anumit client intr-o anumita coada precum si iesirea unui client dintr-o coada.

2.2 Obiective secundare:

Obiectiv Secundar	Descriere	Capitol
Alegerea structurilor de date	Specificarea structurilor de date folosite pentru indeplinirea cerintelor descrise mai in sus.	4
Impartirea pe clase	Diagrama de clase, diagram de secventa	4
Dezvoltarea algoritmilor	Algoritmii propusi pentru rezolvarea logicii a acestei aplicatii.	4
Implementarea solutiei	Descrierea fiecarei clase in parte, cu metode aferente ei, inclusive descrierea interfetei utilizator.	5
Testare	Imagini cu diferite teste desfasurate asupra aplicatiei.	6

3 Analiza problemei

La o analiză sumară a cerinței temei curente, complexitatea ei pare una redusă, implicând o simulare a unui ansamblu de cozi, in care o noua entitate care începe folosirea cozii este plasata la o coada. Insa pentru realizarea acestui lucru intr-un mod cat mai optim este nevoie de mai multe fire de executie. De aceea, se pune accentul pe consolidarea cunoștințelor despre posibilitatea de utilizare a thred-urilor, cât și pe implementarea propriu-zisă într-un limbaj de programare orientat pe obiecte.

Reprezentarea cozilor in formatul intern al programului se realizeaza printr-o tehnica Divide et Impera, intrucat operatiile nu pot sa se realizeze pe o sectiune contigua de date, ci doar daca coada se divide in subparti si asupra acestor subparti se pot aplica operatiile necesare, cum ar fi determinarea timpilor, asezarea in coada respectiva.

4 Proiectare

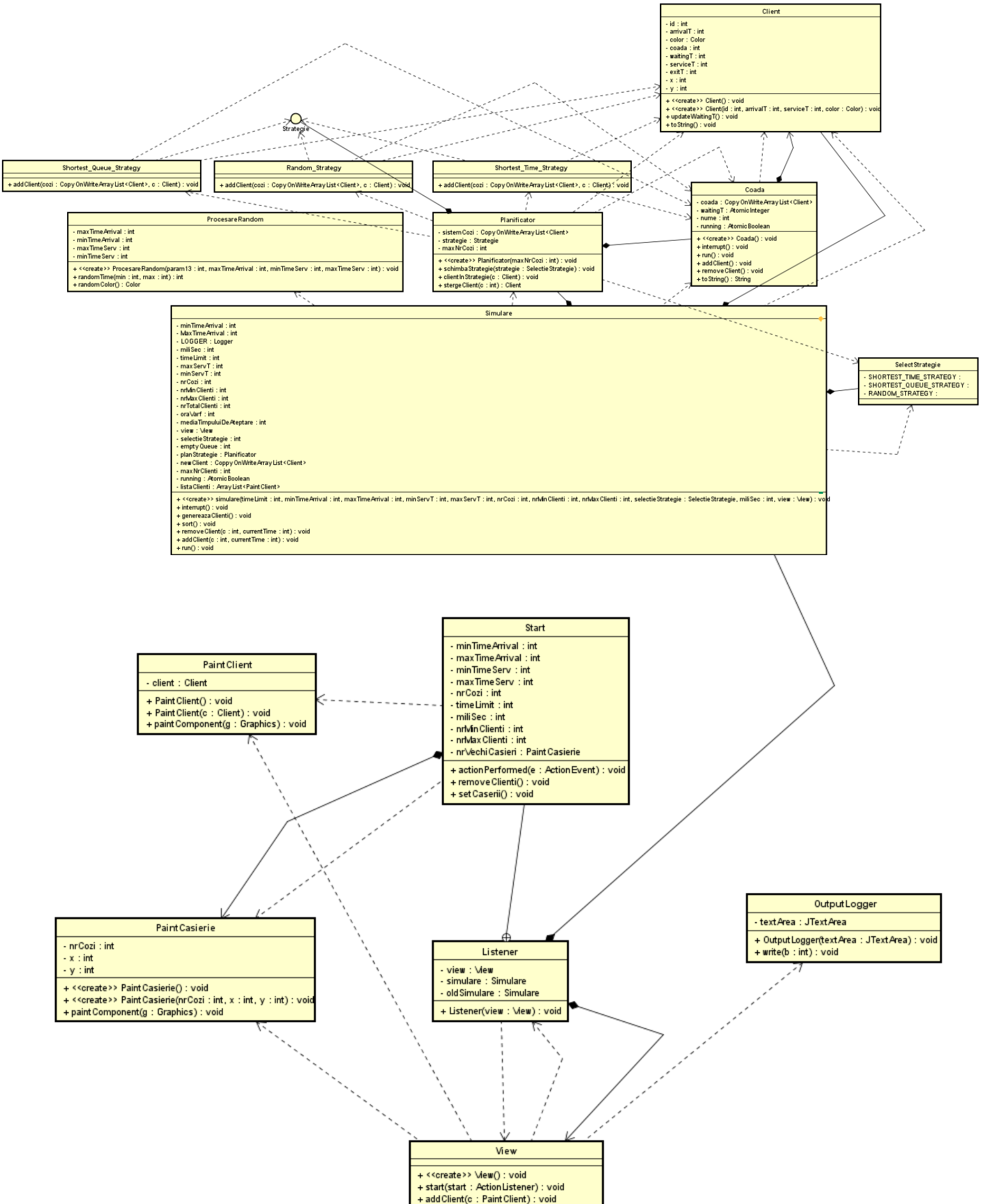
4.1 Structuri de date

Structurile de date care s-au utilizat sunt : CopyOnWriteArrayList si Atomic – folosite pentru sincronizarea thread-urilor.

4.2 Diagrama de clase

O etapa importanta in procesul de proiectare al unei aplicații este realizarea diagramelor UML (Unified Modelling Language) care permit specificarea structurii proiectului, a variabilelor, metodelor, cat si a interacțiunii dintre clase.

Diagrama de clasa este utilizata in descrierea structurii statice, adică a entităților sau claselor existente intr-un sistem. Diagrama de clase reflecta diferite legături intre entitățile domeniului de obiecte si descriu structura interna si vizibilitatea lor (publice, private, protejate) precum si tipurile de relații. Diagrama de clasa este reprezentata prin următoarea schema:


Client
- id : int
- arrivalT : int
- color : Color
- coada : int
- waitingT : int
- serviceT : int
- exitT : int
- x : int
- y : int
+ <<create>> Client() : void
+ <<create>> Client(id : int, arrivalT : int, serviceT : int, color : Color) : void
+ updateWaitingTQ : void
+ toString() : void
Strategie
Shortest_Queue_Strategy
+ addClient(cozi : CopyOnWriteArrayList<Client>, c : Client) : void
Random_Strategy
+ addClient(cozi : CopyOnWriteArrayList<Client>, c : Client) : void
Shortest_Time_Strategy
+ addClient(cozi : CopyOnWriteArrayList<Client>, c : Client) : void
Coada
- coada : CopyOnWriteArrayList<Client>
- waitingT : AtomicInteger
- nume : int
- running : AtomicBoolean
+ <<create>> Coada() : void
+ interrupt() : void
+ run() : void
+ addClient() : void
+ removeClient() : void
+ toString() : String
ProcesareRandom
- maxTimeArrival : int
- minTimeArrival : int
- maxTimeServ : int
- minTimeServ : int
+ <<create>> ProcesareRandom(param13 : int, maxTimeArrival : int, minTimeServ : int, maxTimeServ : int) : void
+ randomTime(min : int, max : int) : int
+ randomColor() : Color
Planificator
- sistemCozi : CopyOnWriteArrayList<Client>
- strategie : Strategie
- maxNrCozi : int
+ <<create>> Planificator(maxNrCozi : int) : void
+ schimbaStrategie(strategie : SelectieStrategie) : void
+ clientInStrategie(c : Client) : void
+ stergeClient(c : int) : Client
Simulare
- minTimeArrival : int
- MaxTimeArrival : int
- LOGGER : Logger
- milliSec : int
- timeLimit : int
- maxServT : int
- minServT : int
- nrCozi : int
- nrMinClienti : int
- nrMaxClienti : int
- nrTotalClienti : int
- oraVarf : int
- mediaTimpuluiDeAsteptare : int
- view : View
- selectieStrategie : int
- emptyQueue : int
- planStrategie : Planificator
- newClient : CoppyOnWriteArrayList<Client>
- maxNrClienti : int
- running : AtomicBoolean
- listaClienti : ArrayList<PaintClient>
+ <<create>> simulare(timeLimit : int, minTimeArrival : int, maxTimeArrival : int, minServT : int, maxServT : int, nrCozi : int, nrMinClienti : int, nrMaxClienti : int, selectieStrategie : SelectieStrategie, milliSec : int, view : View) : void
+ interrupt() : void
+ genereazaClienti() : void
+ sort() : void
+ removeClient(c : int, currentTime : int) : void
+ addClient(c : int, currentTime : int) : void
+ run() : void
SelectStrategie
- SHORTEST_TIME_STRATEGY :
- SHORTEST_QUEUE_STRATEGY :
- RANDOM_STRATEGY :
Start
- minTimeArrival : int
- maxTimeArrival : int
- minTimeServ : int
- maxTimeServ : int
- nrCozi : int
- timeLimit : int
- milliSec : int
- nrMinClienti : int
- nrMaxClienti : int
- nrVechiCasieri : PaintCasierie
+ actionPerformed(e : ActionEvent) : void
+ removeClienti() : void
+ setCaserii() : void
PaintClient
- client : Client
+ PaintClient() : void
+ PaintClient(c : Client) : void
+ paintComponent(g : Graphics) : void
PaintCasierie
- nrCozi : int
- x : int
- y : int
+ <<create>> PaintCasierie() : void
+ <<create>> PaintCasierie(nrCozi : int, x : int, y : int) : void
+ paintComponent(g : Graphics) : void
Listener
- view : View
- simulare : Simulare
- oldSimulare : Simulare
+ Listener(view : View) : void
OutputLogger
- textArea : JTextArea
+ OutputLogger(textArea : JTextArea) : void
+ write(b : int) : void
View
+ <<create>> View() : void
+ start(start : ActionListener) : void
+ addClient(c : PaintClient) : void

4.3 Algoritmi

Algoritmii pe care i-am utilizat sunt de natura in principal logica, dintre acestia putem amintii adaugarea unui client in coada prin verificarea strategiilor, algoritmul de determinare a strategiilor de alegere a cozii respective, stergerea unui client dintr-o coada, crearea unui timp real in care sa functioneze aplicatia.

5 Implementare

Pentru implementarea aplicatei au fost folosite 8 clase, 1 interfata si 1 enumeratie. De asemenea s-au folosit inca 5 clase pentru interfata grafica. Aceste clase sunt descrise mai in jos:

5.1 Clasa Client

Aceasta clasa are varibile de instanta precum: id clientului, timpul de sosire a clientului in coada, timpul de iesire, timpul de prelucrare, timpul de asteptare in coada, coada in care se afla. Cum este solicitata o funcționalitate a aplicației asemănătoare cu sistemul din lumea reala, atunci este implementat următorul principiu: când clientul ajunge in poziția din coada in care urmează sa fie servit, timpul sau de așteptare scade treptat, cu cate o secunda. Atunci când timpul de așteptare devine 0, clientul va părăsi coada. Deoarece clientii trebuie reprezentati dinamic intr-o interfata grafica, am retinut pozitia in sistemul de coordonate xOy in care se afla. Pentru o reprezentare intuitiva, dar si atractiva, se retine pentru fiecare client o culoare, care va fi generata aleator. Astfel, se permite distingerea intre clienții aceleiași cozi, dar si intre clienți din cozi diferite. De asemenea sunt implementare metode de setare si de preluare a datelor despre un client cum ar fi: getExitT(), setExitT(), getArrivalT(),

setArrivalT(), getId(), getCoada(), getWaitingT(), updateWaitingT(), getX(), setX(), getY(), setY(), getColor() etc.

5.2 Clasa Coada

Aceasta implementeaza interfata Runnable si suprascrie metoda run(). Cu ajutorul acestei metode definim modul de lucru al firelor de executie care sunt impartite pentru fiecare coada in parte. Astfel, in momentul in care un client vine la coada el este preluat de catre un fir de executie, iar in momentul cand acesta este procesat este folosita metoda sleep() pentru ca thread-ul sa nu fie intrerupt de catre un alt thread in timp ce acesta proceseaza clientul. De asememenea in metoda run() se incrementeaza si timpul de asteptare total la o coada, incrementandu-se valoarea cu timpul de procesare a clientului si decrementandu-se dupa ce un client a fost procesat si iese din coada. Pe langa aceasta metoda care are rol de sincronizare a thread-urilor, mai sunt definite si implementate alte metode cum ar fi: addClient() care adauga un client in coada, creste timpul de asteptare cat si numarul de clienti la acea coada, deleteClient() cu rol de stergere a unui client din coada si de decrementare a timpului de asteptare al unui client la o coada, precum si de modificarea pozitiilor in sistemul xOy al celorlalti clienti ramasi in coada. Pe langa acestea se mai afla metodele accesoare si de setare a variabilelor instanta. Aceasta clasa va folosi o colectie pentru stocarea clientilor, un exemplu elocvent este CopyOnWriteArrayList care este Thread Safety. Alte variabile instanta sunt: name- numele/indexul cozii, waitingT de tip AtomicInteger reprezentand tipul de asteptare al unui client la o coada, precum si o variabila running care ne spune daca threadul ruleaza fiind setata pe true iar in momentul cand acesta este intrerupt aceasta va fii setata pe false.

5.3 Clasa Planificator

Aceasta detine ca si variabile instanta o lista de cozi de tip CopyOnWriteArrayList, variabila ce implementeaza interfata de tip Strategie, cu rol de a selecta strategia de adaugare a unui client intr-o coada. Ca si metode se remarca schimbaStrategia() cu rolul de a seta strateia de adaugare a unui client intr-o coada si clientInStrategie() cu rolul de a adauga acel client intr-o coada in functie de strategia aleasa, stergeClient() care sterge un anumit client din coada in care se afla. Pe langa aceste metode se mai regasesc metodele accesoare si de setare a variabilelor instanta specifice fiecarei clase in parte.

Clasa ProcesareRandom care contine urmatoarele variabile instanta: timpul minim si maxim de prelucrare a clientilor, timpul minim si maxim de sosire a clientilor. Metoda randomTimel() genereaza un numar aleator din intervalul

respectiv pentru timp si sunt folosite in clasa Simulare pentru a genera aleator clientii care sunt adaugati in sistemul de tip coada. Se mai remarca metoda randomColor() care genereaza aleator o culoare pentru fiecare client in parte.

5.4 SelectieStrategie

Enumeratia SelectieStrategie defineste trei constante cu scopul de facilitare in gasirea strategiei potrivite introduse de catre utilizator. Astfel evem: **SHORTEST_TIME_STRATEGY**, **SHORTEST_QUEUE_STRATEGY** si **RANDOM_STRATEGY.**

5.5 Strategie

Interfata Strategie propune implementarea diferita a metodei addClient() de catre clasele care implementeaza interfata respectiva. Aceasta metoda adauga clientul sosit intr-o coada in functie de cerintele specificate in interfata utilizator.

5.6 Shortest_Time_Strategy

Clasa Shortest_Time_Strategy implementeaza interfata Strategie si defineste metoda specifica interfetei, addClient(). Aceasta metoda adauga clientul sosit intr-o coada in functie de timpul de asteptare, acesta trebuind sa fie cel mai mic timp de asteptare dintre toate cozile. De asemenea se adauga si pozitia clientului in coada.

5.7 Shortest_Queue_Strategy

Clasa Shortest_Queue_Strategy implementeaza interfata Strategie si defineste metoda a carei semnatura este descrisa in interfata, addClient(). Aceasta metoda adauga clientul sosit, intr-o coada , in functie de marimea cozilor repsective, alegandu-se coada cu cel mai mic numar de clienti. De asemenea se adauga si pozitia clientului in coada.

5.8 Random_Strategy

Clasa Random_Strategy implementeaza interfata Strategie si defineste metoda specifica interfetei, addClient(). Aceasta metoda adauga clientul sosit intr-o coada random. De asemenea se adauga si pozitia clientului in coada.

5.9 SimulationShop

Clasa SimulationShop implementeaza interfata Runnable si are rol de clasa coordonatoare a intregii activitati desfasurate in aplicatie. Aceasta clasa va gestiona numărul de cozi, intervalul de timp in care sosesc clienții, cat de des sosește un client, dar si timpul de servire al fiecărui client. In constructor sunt instantiate variabilele, urmand ca logica sa fie decrisa in metoda run(). Aici este definita o variabila de tip integer, currentTime cu rolul de a se afla fiecare moment discret de timp al intervalului de simulare. La fiecare unitate de timp se parcurge lista generata de clienti , cei care au finalTime egal cu currentTime sunt eliminati din lista, iar cei care au arrivalTime egal cu currentTime sunt adaugati in lista. Tot in aceasta metoda sunt mereu actualizate valorile pntru variabilele oraVarf si mediaTimpuluiDeAsteptare. De asemenea, este actualizata interfata grafica de fiecare data cand are loc o modificare a clientilor. Secvența de instrucțiuni care generează un client este executata in intervalul de timp specificat de către utilizator (interval furnizat prin intermediul interfeței grafice). Metoda generezaClienti() creează un nou obiect de tip *Client*, care este inițializat cu un număr de identificare unic, apoi ii este asignat timpul necesar efectuării serviciului solicitat si timpul sosorii (timpi generati aleator in intervalul specificat de utilizator). De asemenea, tot aleator, fiecărui client ii este asignata o culoare specifica, pentru a putea fi reprezentat si in interfața grafica. Clientul este adăugat in variabila coada urmand ca sa fie procesat si sa fie atribuit cozii specifice in funtie de strategia aleasa de utilizator.

5.10 Interfata

Graphical User Interface este o interfață cu utilizatorul bazată pe un sistem de afișaj ce utilizează elemente grafice. Interfața grafică este numit sistemul de afișaj grafic-vizual pe un ecran, situat funcțional între utilizator și dispozitive electronice. Folosim o interfata grafica User-Friendly cu scopul de a putea fi folosit acest simulator de cozi.

Clasa View reprezinta interfata grafica, care folosește elementele puse la dispoziție de limbajul Java. Aceasta conține elementele constituente ale interfeței grafice propriu-zise. Variabilele de clasă vor fi obiecte specifice interfeței grafice (precum casete text, butoane), iar constructorul clasei va

inițializa și așeza aceste elemente în designul dorit. Ca și metode, vor fi necesare metode de adăugare a "ascultătorilor de acțiuni" (*ActionListener*) care vor monitoriza interacțiunile utilizatorului cu programul, și vor obține datele introduse de utilizator, date care vor fi procesate în concordanță cu opțiunea dorită de utilizator, iar apoi rezultatele vor fi afișate într-un mod adecvat.

Clasa Listener conține o subclasa care implementează ascultătorul care detectează activarea secvenței de pornire a modelului. Aceasta subclasa primește de la interfața grafică datele introduse de utilizator apoi le transmite clasei care realizează efectiv aceste operații. Clasa ce conține operațiile trimite înapoi rezultatele aferente operațiilor, care sunt interpretate de către unitatea de control, iar datele sunt transmise interfeței grafice sub formă de text, numere, sau elemente grafice, care vor fi afișate utilizatorului. Subclasa are un mecanism de tratare a excepțiilor, excepția principală fiind excepția care apare în urma unor date de intrare greșite, caz în care, un mesaj adecvat va fi transmis utilizatorului, permițându-i eliminarea erorilor de introducere a datelor.

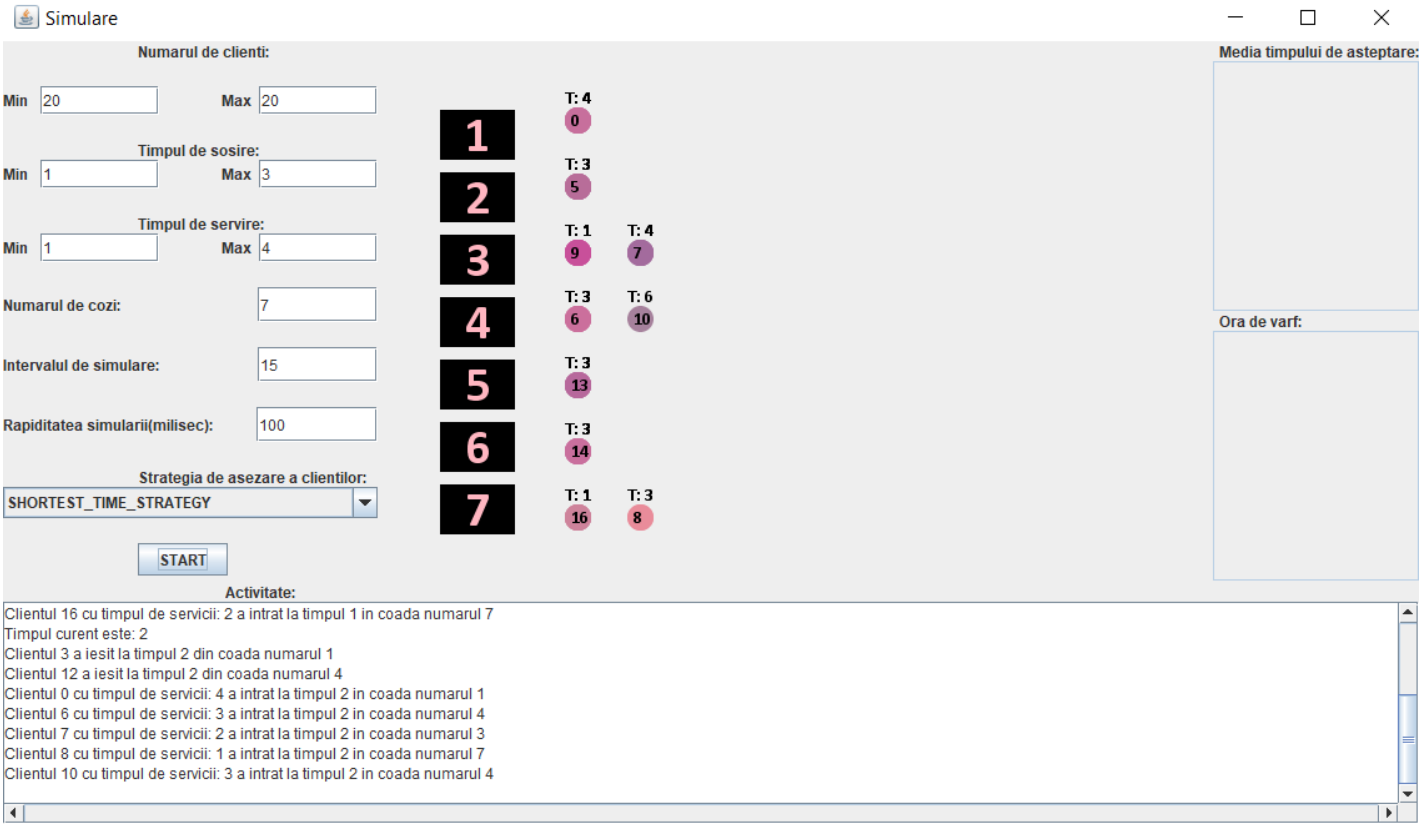
Clasa OutputLogger care extinde clasa OutputStream supsracrie metoda write din clasa parinte cu rolul de a redirectiona consola intr-un textArea din interfata grafica.

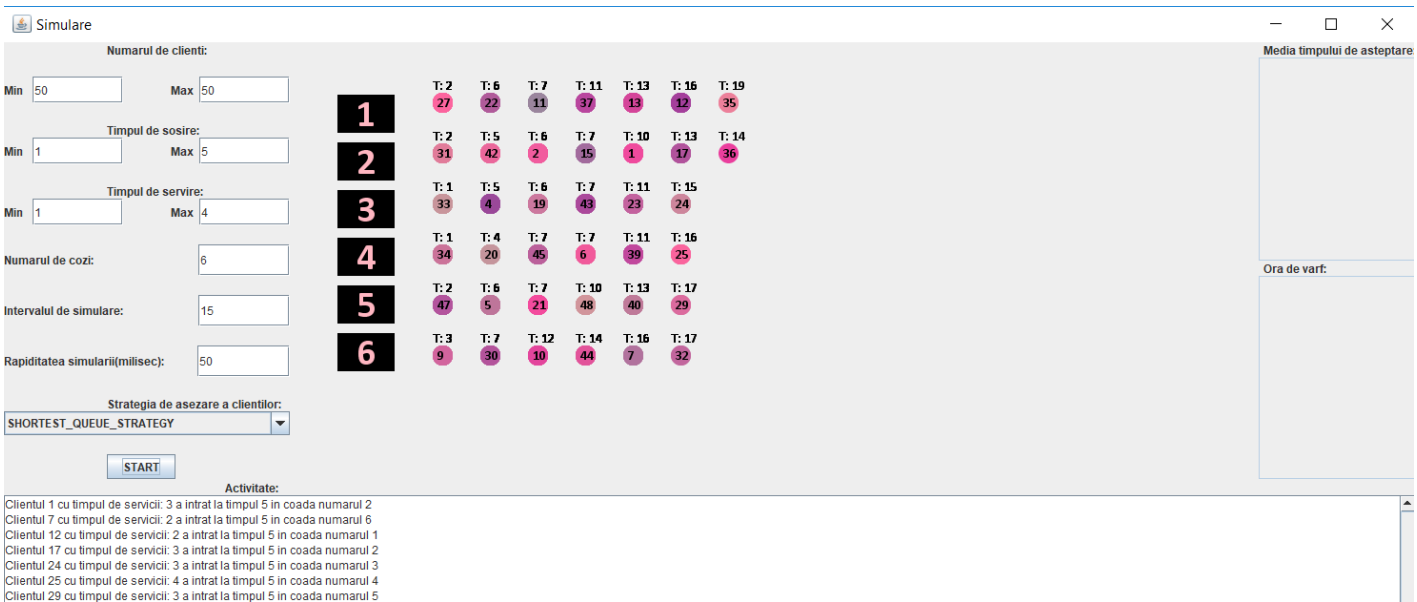
Clasa PaintCasierie extinde clasa JComponent realizand un desen grafic care reprezinta casieriile, ce urmeaza sa fie afisate in interfata grafica, corespunzandu-i fiecarei cozi un desen.

Clasa PaintClient extinde clasa JComponent realizand un desen grafic care reprezinta clientul, care urmeaza sa fie afisata in interfata grafica, corespunzandu-i fiecarei client un desen.

6 Testare

In imaginile de mai in jos este prezentata interfata aplicatiei:





7 Dezvoltari ulterioare

Aplicația curentă oferă utilizatorului posibilitatea de a porni si asista simularea unui sistem de cozi, prin intermediul unei interfețe grafice simplificate. Această implementare poate fi dezvoltată și extinsă, pornind de la mai multe considerente.

Extinderea datelor de intrare poate constitui o sursa consistenta pentru creșterea utilității aplicației. Astfel, se pot adaugă diferite condiții asupra cozilor, precum timpi de pauza, alternarea cozilor, relocarea clienților.

Adăugarea de sunete relevante poate face aplicația sa fie mult mai practica, clienții fiind anunțați in prealabil despre deschiderea sau închiderea unui post in care se îndeplinesc serviciile cerute.

In plus, diferiții parametri ai aplicației ar putea fi modificați chiar in timpul rulării, acest lucru necesitând o serie de modificări care sa permită aplicației sa reacționeze in timp real la datele de intrare, si sa ofere răspunsuri in concordanta cu nevoile utilizatorului.

De asemenea, interfața grafică poate fi îmbunătățită pentru a fi mai intuitivă și mai atractivă pentru orice tip de utilizator. Fie că este vorba de persoane care pot lucra într-un mod facil cu calculatoarele, sau, în opoziție, persoane pentru care interacțiunea cu un astfel de program nu este atât de accesibilă, sistemul trebuie să aibă o interfață grafică solidă, o grupare logică și simplă a elementelor de aceeași natură, și să ofere acces imediat la toate funcțiile și posibilitățile puse la dispoziție de către program.

8 Bibliografie

Giosan Ion, POO curs 6 - Dezvoltarea aplicațiilor OO. Diagrame UML de clase și obiecte, UTCN, 2017

Giosan Ion, POO curs 9 - Interfete utilizator grafice (GUIs), UTCN, 2017

http://www.coned.utcluj.ro/~salomie/PT_Lic/

http://coned.utcluj.ro/~marcel99/PT/Tema%202/Java%20Concurrency.pdf

https://www.tutorialspoint.com/uml/index.htm

http://www.uml-diagrams.org/

https://stackoverflow.com/