MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICA SI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

DISCIPLINA TEHNICI DE PROGRAMARE

Documentatie Tema2

- Simulare cozi magazin -

Popa Alexandra Maria

Grupa 30224

CTI-ro

Cuprins:

1.Obiectivul Temei :

Enuntul Temei: Propunerea, proiectarea si implementarea unui sistem de simulare a evolutiei cozilor unui magazin in timp real.Se urmareste reducerea timpului de asteptare prin eficentizarea metodelor de introducere a clientilor in coada. De asemenea, proiectul trebuie sa includa o interfata grafica pentru preluarea datelor de intrare si afisarea in timp real a datelor rezultate in urma procesului de simulare a cozilor. Evolutia cozilor va fi reprezentata prin intermediul unui camp de text ce afiseaza informatiile jurnalizate in urma actiuniilor ce au fost efectuate la un anumit moment de timp in cozi.

Obiective :

Obiectiv principal : Obiectivul acestei teme este proiectarea si implementarea unei aplicatii de simulare care are in vedere analiza sistemelor bazate pe asteptare la coada pentru determinarea si minimizarea timpului de asteptare al clientilor.

Cozile sunt frecvent utilizate pentru a modela domenii din lumea reală. Gestionarea sistemelor bazate pe coada de asteptare au ca scop reducerea timpului de asteptare in cozi a clientilor, inainte de a fi serviti. O modalitate de a minimiza timpul de asteptare este de a adauga mai multe servere, adica mai multe cozi in sistem. Fiecare coada este considerata ca avand un asociat procesor. Insa aceasta abordare mareste costurile furnizorului de servicii. Cand un nou server este adaugat, clientii vor fi distribuiti în mod egal tuturor cozilor curente disponibile.

Aplicația ar trebui sa simuleze o serie de clienti care sosesc pentru anumie servicii, sunt introdusi in cozi, pusi sa astepte, ca mai apoi sa fie serviti sis a paraseasca in final coada. Este monitorizat timpul petrecut de client in coada si timpul mediu de asteptare. Pentru a calcula timpul de asteptare este nevoie sa cunoasca ora sosirii, timpul de finalizare a serviciilor dorite și timpul de procesare. Timpul de sosire si timpul de procesare depind de fiecare client in parte - cand ajung in coada si de de cat timp au nevoie pentru finalizarea serviciilor dorite. Timpul de iesire din coada depinde de

numarul de cozi, de numărul de clienti din coada și de timpul necesar pentru realizarea fiecarui serviciu in parte .

Obiective secundare:

- organizarea pe clase -> utilizarea diagramelor de clase
- dezvoltarea algoritmilor folosind POO -> algoritmi de sincronizare si procesare a clientilor implicati in simularea functionarii unei cozi dintr-un magazin
- utilizarea structurilor de date -> specific pentru lucrul cu diferiti algoritmi ce au dus la realizarea simularii
- formularea de scenarii -> modalitati de preluare si introducere a clientilor in cozi urmarind un anumit timp si o anumita logica
- implementarea si testarea solutiilor obtinute -> testare prin jurnalizarea si afisarea in interfata grafica a pasilor de simulare a cozilor ce se proceseaza in timp real .

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

Problema enuntata necesita cunostiinte leagate de lurcrul cu fire de executie (thread-uri) , gestionarea acestore in functie de un anumit timp curent, pentru realizare simularii in timp real al cozilor , cat si cunoasterea programarii orientate pe obiect pentru crearea si manipularea obiectelor. Totodata, este nevoie de cunoasterea modalitatilor de crearea a interfetei grafice si de afisarea a mesajelor din program in timp real prin intermediul unei ferestre de simulare.

Din analiza problemei rezulta faptul ca avem nevoie de un set de date de intrare si un set de date de iesire. Datele de intrare sunt reprezentate de numarul de clienti din magazine , numarul de cozi disponibile , un timp de sorire minim , un timp de sosire maxim , un timp de procesare minim , un timp de procesarea maxim cat si timpul in care sa se faca simularea cozilor. Metoda de introducere a datelor a fost special aleasa pentru ca utilizatorul care face simularea sa poata alege intervalele de timp si

numarul de resurse (client si cozi) in funcie de preferinta proprie si in functie de ce doreste sa urmareasca odata cu simularea.

Dupa introducere datelor de intrare se incepe simularea prin apasarea unui buton de start. In urma acestei actiuni sunt preluate datele de intrare date de la tastatura si astfel se formeaza sablonul specific de simulare, astfel : clientii sosesc in coada la anumite intervale de timp , fiecare avand un anumit timp de sosire primit in mod aleator. Acestia sosesc cu scopul indeplinirii unor servicii care au de asemenea un timp de realizarea care variaza de la un client la altul. Trebuie sa se ia decizia de introducere a clientilor intr-o anumita coada. Clientii sunt plasati in cozi dupa un algoritm care alege pentru un anumit client sosit la un moment dat , coada cu cel mai mic timp de asteptare. Dupa introducerea clientului in coada, acesta trebuie sa astepte un anumit interval de timp reprezentat de timpul de procesarea al clientilor din fata lui. Dupa terminarea timpului de asteptare, clientul este servit si paraseste coada.

Evolutia simularii cozilor este ilustrara prin intermediul afisarii cozilor la fiecare timp curent din timpul total de simulare . Aceasta afisare se realizaza in interfata grafica si este vizibila utilizatorului. Aceasta metoda permite observarea si analizarea modificarilor produse in timp si creaza o interactiune mult mai buna a utilizatorul cu aplicatia. Afisarea trebuie sa fie clara pentru utilizator, scopul fiind ca produsul final sa fi cat mai practic si usor de folosit.

3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator) :

Proiectarea aplicatiei de simulare a cozilor a necesitat proiectarea a 6 clase *: Client , Queue , Manager , Shop , Generator , ViewPrincipal , Controller , CustomOutputStream* si *App.* Interfata grafica se bazeaza pe sablonul MVC ce presupune separarea proiectului in Model, View si Controller, in functie de nevoi, reliefand astfel principiile POO.In continuare voi prezenta proiectarea in mare a claselor ce formeaza aplicatia, metodele acestora fiind descrise mai pe larg in capitolul 4.

Client -> aceasta clasa are patru variabile instanta : *id* – unic pentru fiecare client ce soseste in magazine; *arrivedTime* – timpul de sosire in coada; *serviceTime* –timpul necesar de indeplinire a serviciilor dorite; *exitTime*- timpul la care clientul paraseste coada. Toate aceste variabile instanta sunt de tip *int* . Clasa contine metode de tip setter si getter: *getId() , getArrivedTime() , getServiceTime() , setId(int id) , setArrivedTime(int arrivedTime) , setServiceTime(int serviceTime) , getExitTime() , setExitTime(int exitTime)* si implementeaza metoda *compareTo(Client c)* si *toString( )*.

Queue -> contine 4 variabile instanta : *client* – reprezinta o lista de tipul *BlockingQueue<Client>* ce stoceaza clientii astfel incat sa poata sa fie accesati la un moment dat doar de o un singur fir de executie; *totalTime* – de tipul *AtomicInteger* – este tipul total de servire de pe o coada; *si LOGGER* – variabila utilizata pentru jurnalizare; *text* -de tipul *JTextField*, utilizata pentru afisarea cozilor in interfata grafica. Metodele utilizate sunt : *getClients() , setClients(BlockingQueue<Client> clients) , getTotalTime() , setTotalTime(AtomicInteger totalTime) , metoda de run( ), addClient(Client c)* si *listQueue( ).*

Manager -> contine ca variabila instanta *listOfQueue* de tipul *List<Queue>*, ce reprezinta o lista in care se stocheaza toate cozile functionale din magazin. De asemenea, variabila instanta LOGGER se foloseste pentru jurnalizare. Clasa contine metoda *addClientInRightQueue(Client c)* si un getter si setter :*getListOfQueue( ) , setListOfQueue( )* .

Generator -> are cinci variabile instant de tipul int : *nrClients* – numarul de clienti ale caror date trebuie generate aleator si care trebuie introdusi intr-o list ace reprezinta totalitatea clientilor ca au intrat in magazine ; *minIntervalOfArrivingTime , maxIntervalOfArrivingTime , minIntervalOfServingTime , maxIntervalOfServingTime* , ce reprezinta intervalele de timp introduse de catre utilizator si care contribuie la generarea aleatoarea a timpilor ce vor fi atribuiti fiecarui client. Clasa contine metode de tip getter / setter : *getMinIntervalOfArrivigTime( ) , getMaxIntervalOfArrivigTime( ) , getMinIntervalOfServiceTime( ) , getMaxIntervalOfServiceTime( ) , setMinIntervalOfArrivigTime(int*

minIntervalOfArrivigTime) , setMaxIntervalOfArrivigTime(int maxIntervalOfArrivigTime) , setMinIntervalOfServiceTime(int minIntervalOfServiceTime) , setMaxIntervalOfServiceTime(int maxIntervalOfServiceTime) , dar si metode precum : *generateArrivalTime( ), generateServiceTime( ) , generateNrClients( ).*

Shop -> reprezinta clasa care simuleaza magazinul. Aceasta clasa contine 8 variabile instanta de tip int ,ce reprezinta informatiile introduse de catre utilizoator de la tastatura : *nrClients , nrQueue ,currentTime, simulationTime, minIntervalOfArrivingTime , maxIntervalOfArrivingTime , minIntervalOfServingTime si maxIntervalOfServingTime*. Clasa implementeaza metoda *run( )* interiorul careia se descrie logica din spatele controlului asupra cozilor din magazin.

Clasele prezentate mai sus reprezinta modelul aplicatiei , deoarece acestea definesc logica din spatele procesarii datelor de intrare.

ViewPrincipal -> reprezinta interfata grafica a aplicatiei; prin intermediul elementelor care o compun se creaza legatura cu utilizatorul, usurand astfel considerabil modul de utilizare a aplicatiei si vizualizarea rezultatelor obtinute . Pentru realizarea interfetei grafice am folosit 19 variabile instanta, dintre care : 12 campuri de tipul JTextField,7 dintre acestea editabile pentru introducerea datelor necesare simularii (*nrClientsF, nrQueueF, minArrivingTimeF, maxArrivingTimeF, minServiceTimeF, maxServiceTimeF si timpSimulareF*) si 5 needitabile pentru afisarea evolutiei cozilor*(queue1, queue2, queue3, queue3, queue4*) ;13 etichete de tip JLabel care indica semnificatia campurilor text (*labelOptiuni, labelNrC, labelNrQ, labelMinAT, labelMaxAT, labelMinST, labelMaxST, labelTimpSimulare, labelQ1, labelQ1, labelQ1, labelQ1, labelQ1*) ; un buton de tipul *JButton* (*startBtn*) la apasarea caruia se porneste simularea cozilor .Constructorul clasei contine declararea a 12 panel-uri de tip JPanel, in care au fost adaugate variabilele instanta ce compun fereasta de vizualizare. Acesta sunt organizate in asa fel incat sa se pastreze o anumita aliniere a elementelor in feriasta in functie de axele Ox si Oy. Totodata in acest constructor se seteaza vizibilitatea ferestrei principale si numele acesteia. Clasa contine de asemenea si metode de tip getter si

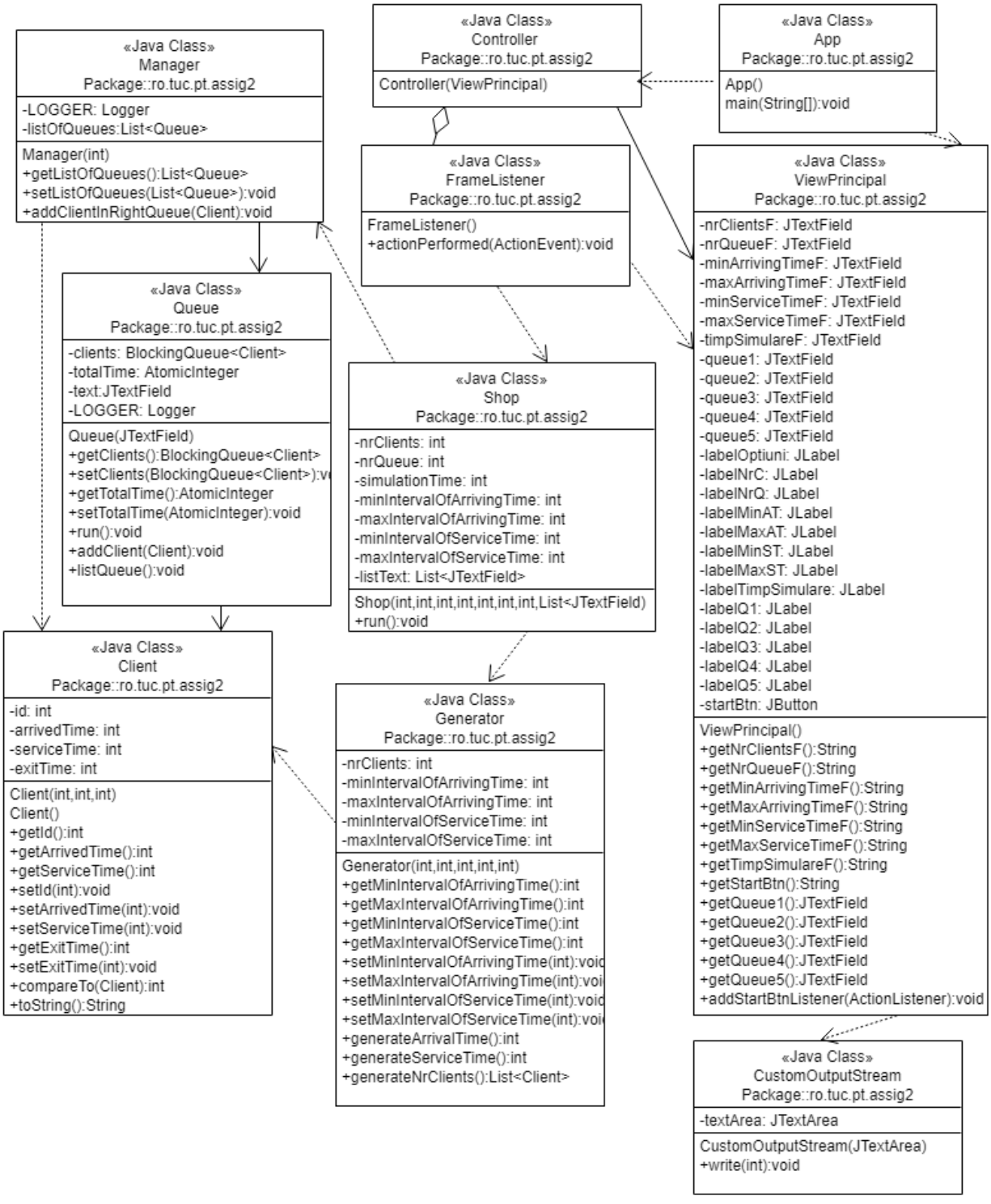
setter care preiau informatia introdusa de catre utilizator in campurile *nrClientsF, nrQueuesF,* minArrivingTimeF, maxArrivingTimeF, minServiceTimeF, maxServiceTimeF si timpSimulareF si o transmit mai departe, si seteaza rezultatul obtinut in urma procesarii datelor de intrare (*getNrClientsF( ) , getNrQueuesF( ) ,getMinArrivingTimeF( ) , getMaxArrivingTimeF( ) , getMinServiceTimeF( ) , getMaxServiceTimeF( ) si getTimpSimulareF( ), getQueue1( ), getQueue2( ), getQueue3( ), getQueue4( ), getQueue5( )*). Pe langa aceste, se gaseste si o metoda care adauga ascultatori (action listeners) pentru butonul de start al simularii *addStartBtnListener(ActionListener l)* .

Controller-> reprezinta partea de control a proiectului, ce decide ce pasi urmeaza sa faca modelul. Clasa contine o singura variabila instanta- *view*, de tipul ViewPrincipal , ce realizeaza legatura cu clasa *ViewPrincipal.* Constructorul clasei contine un ascultator pentru butonul de start , care este adaugat la view. Clasa de control contine o clasa interna *FrameListener* ce implementeaza *ActionListener* si contine metoda *actionPerformed(ActionEvent e)* .

App -> contine programul principal care initializeaza interfata si leaga componentele impreuna.

De asemenea, structurile de date folosite in acest proiect sunt : ArrayList si LinkedBlockingQueue .

In continuare este prezentata diagrama UML de clase a proiectului:



4. Implementare :

In acest capitol se vor descrie deciziile de implementare ale metodelor fiecarei clase in parte, impreuna cu algoritmii ce sunt folositi in realizarea acestora .

Client :

- *compareTo(Client c)* -> metoda implementata cu scopul de a se putea realiza sortarea clientilor in ordine crescatoare in functie de timpul de sosire a acestora.

Queue :

- *addClient(Client c)* -> metoda de adugare propriu-zisa a clientilor in coada si setare a exitTime-ului si a tipului de servire total de pe intreaga coada.
- *run( )* -> metoda suprascrisa in clasa Queue care implementeaza Runnable; metoda proceseaza clinetii ce se afla in coada prin utilizarea threaduri-lor si modalitatilor de sincronizare a acestora. Fiecare coada in parte reprezinta un fir de executie care isi realizeaza independent activitatea fata de celelalte fire de executie .Odata ce clientul a intrat in coada, acesta este preluat si sters, iar mai apoi pentru thread-ul curent se apeleaza metoda *sleep()* ,care il “pune sa doarma” cat timp dureaza servirea clientului procesat. Dupa procesare este scazut timpul de servire a clientului din timul total al cozii si afisat un mesaj care anunta iesirea din coada a clentului respectiv. Aceasta metoda va rula cat timp thread-ul curent este in viata .

Manager :

- *addClientInRightQueue(Client c)* -> metoda utilizata pentru introducerea clientilor in coada potrivita in functie de logica de alegerea a cozi cu timpul de servire total minim, pentru a eficientiza astfel fluxul clientilor care parcurg coada intr-un interval de timp cat

mai scurt. De asemene,metoda afiseaza un mesaj prin care se anunta adaugarea unui anumit client intr-o anumita coada.

Generator :

- *generateArrivalTime( )* –> metoda ce genereaza aleator timpul de sosire al unui cliet in functie de un interval de timp dat de la tastatura.
- *generateServiceTime( )* -> metoda ce genereaza aleator timpul de seervire al unui cliet in functie de un interval de timp dat de la tastatura.
- *generateNrClients( )* -> metoda ce genereaza aleator n clienti si ii introduce intr-o lista, pe care mai apoi o ordoneza in ordine crescatoare in functie de timpul de sosore; din aeasta lista, clientii urmeaza sa fie preluati si introdusi mai departe in cozile potrivite. Metoda returneaza lista clientilor generati.

Shop :

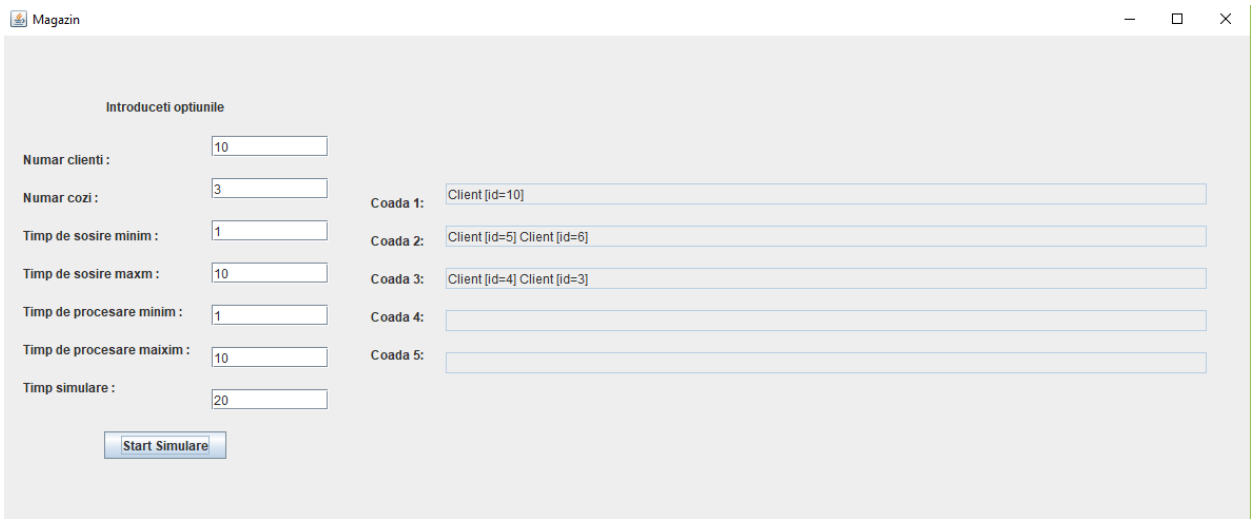
- *run( )* -> metoda ce coordoneaza intregul procesa ce se desfasoara pe parcursul simularii. In functie de un timp curent care creste pana la un anumit timp de simulare introdus in interfata grafica, se preiau clientii din lista ordonata si se verifica daca timpul de sosire al clientului curent este egal cu timpul curent. In caz afirmativ,se apeleaza metoda *addClientInRightQueue(…)* cu ajutorul unui obiect de tip Manager, urmand ca mai departe managerul sa preia controlul si sa procese clientul.

Controller :

- *actionPerformed(ActionEvent e)* => aceasta metoda realizeaza comunicarea cu clasa *ViewPrincipal()*; se preaiu datele de intrare cu care se apeleaza constructorul unui obiect de tip Shop; se instantiaza un obiect de tip Thread (new Thread(Shop)) si mai apui se porneste thread-ul (t.start( )).

5. Rezultate:

Rezultatele obtinute in urma unei simulari sunt prezentate prin intermediul intergetei grafice.



6.Concluzii :

In urma acestei teme am invatat sa imi structurez mai bine codul astfel incat acesta sa respecte pricipiile POO. Am fost pusa in situatia de a realiza o interfata grafica ce afiseaza rezultatele in timp real in JTextFieldf, ceea ce a adus un plus considerabil cunostiitelor legate de lucrul cu modelul MVC, cat si in ceea ce priveste modului de a scrie cod. De asemenea, am dobandit cunostiinte legate de lucrul cu thread-uri si cu metodele de sincronizare ale acestora.

Ca imbunatatiri ulterioare, aplicatia mea ar putea beneficea de adaugarea unei animatii grafice care sa reprezinte clientii si cozile. Totodata, interfata grafica ar putea afisa informatii suplimentare privind timpul mediu de asteptare la coada cat si ora de varf.

7. Bibliografie :

- Indrumator de laborator POO
- Curs POO
- https://stackoverflow.com/
- https://www.baeldung.com/java-wait-notify/