Tehnici de programare

DOCUMENTAȚIE

TEMA 2

-SIMULATOR DE COZI-

Student: Cristian Marius Mureşan
Grupa: 30224
Prof.: Ioan Salomie
Asistent laborator: Călina Cenan

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare (diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)
4. Implementare si testare
5. Rezultate
6. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare
7. Bibliografie

1. Obiectivul temei

Obiectiv: Proiectati si implementati o aplicatie menita sa analizeze si sa simuleze un sistem bazat pe cozi.

Descriere: Cozile sunt foarte des intalnite in lumea reala si in diferite modele. Principalul obiectiv al unei cozi este de a asigura un loc unui client unde sa astepte pentru a i se oferi un serviciu. Managementul sistemelor bazate pe cozi este interesat de minimizarea timpului de asteptare al unui client in coada.O modalitate de minimizare a timpului este de a adauga mai multi operatori, respective mai multe cozi in sistem (considerand ca fiecare coada are propriul processor), dar aceasta abordare mareste costurile. La deschiderea unei noi cozi, clientii care asteapta la celelalte sunt distributi in mod egal in noua coada.

Sistemul trebuie sa simuleze o serie de client care sosesc si cer un serviciu, intrarea in cozi, asteptarea, servirea si, in final, parasirea cozii. Pentru a calcula timpul de asteptare avem nevoie sa stim timpul de sosire, timpul de plecare si timpul alocat serviciului cerut. Timpul de sosire si timpul de servire depinde de fiecare individ in parte. Timpul de plecare depinde de numarul de cozi, de numarul de clienti in coada si de serviciul solicitat de ceilalti clienti.

Date de intrare:

- Intervalul minim si maxim de sosire a noilor client
- Timpul minim si maxim al serviciului solicitat
- Numarul de cozi
- Intervalul de simulare
- Alte informatii considerate necesare

Date minime de iesire:

- Timpul mediu de asteptare al clientilor, de servire, considerant un interval de simulare si un anumit numar de cozi
- Un fisier de iesire cu evenimentele si datele din sistem
- Evolutia cozii
- Ora de varf din decursul simularii

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

Aplicatia trebuie sa simuleze asteptarea unor clienti pentru a beneficia de un serviciu dorit (exemplu: banca, magazin, etc.). Precum in lumea reala, acestia trebuie sa astepta in cozi, iar fiecare coada este total independenta ce celelalte. Ideea este de a analiza cati clienti pot fi serviti intr-un anumit interval orar, care este timpul mediu de asteptare, timpul mediu de servire si ora de varf in functie de anumiti parametri introdusi in interfata utilizatorului.

Scenariul ales pentru dezvoltarea acestei aplicatii este bazat pe un interval orar de activitate, precum in lumea reala, pe decursul caruia clientii sosesc, sunt orientati de fiecare data spre coada cea mai scurta, urmand sa-si astepte randul si a fi serviti. Dupa finalizarea procesului de servire, fiecare client paraseste coada si, totodata, magazinul.

Parametri necesari unei simulari sunt:

- Numarul de cozi ce se deschid in cadrul magazinului - La deschidere cozile sunt goale, iar acestea vor fi populate in functie de sosirea clientilor.
- Intervalul minim si maxim intre 2 clienti consecutivi ce intra in sistem - Acest interval seteaza timpul trecut de la sosirea unui client pana la sosirea urmatorului.
- Intervalul minim si maxim de servire - Acest interval reda timpul pe care fiecare client il are de asteptap pentru a ii fi oferit serviciul
- Intervalul de simulare – timpul de deschidere si cel de inchidere a aplicatie, respectiv a magazinului.

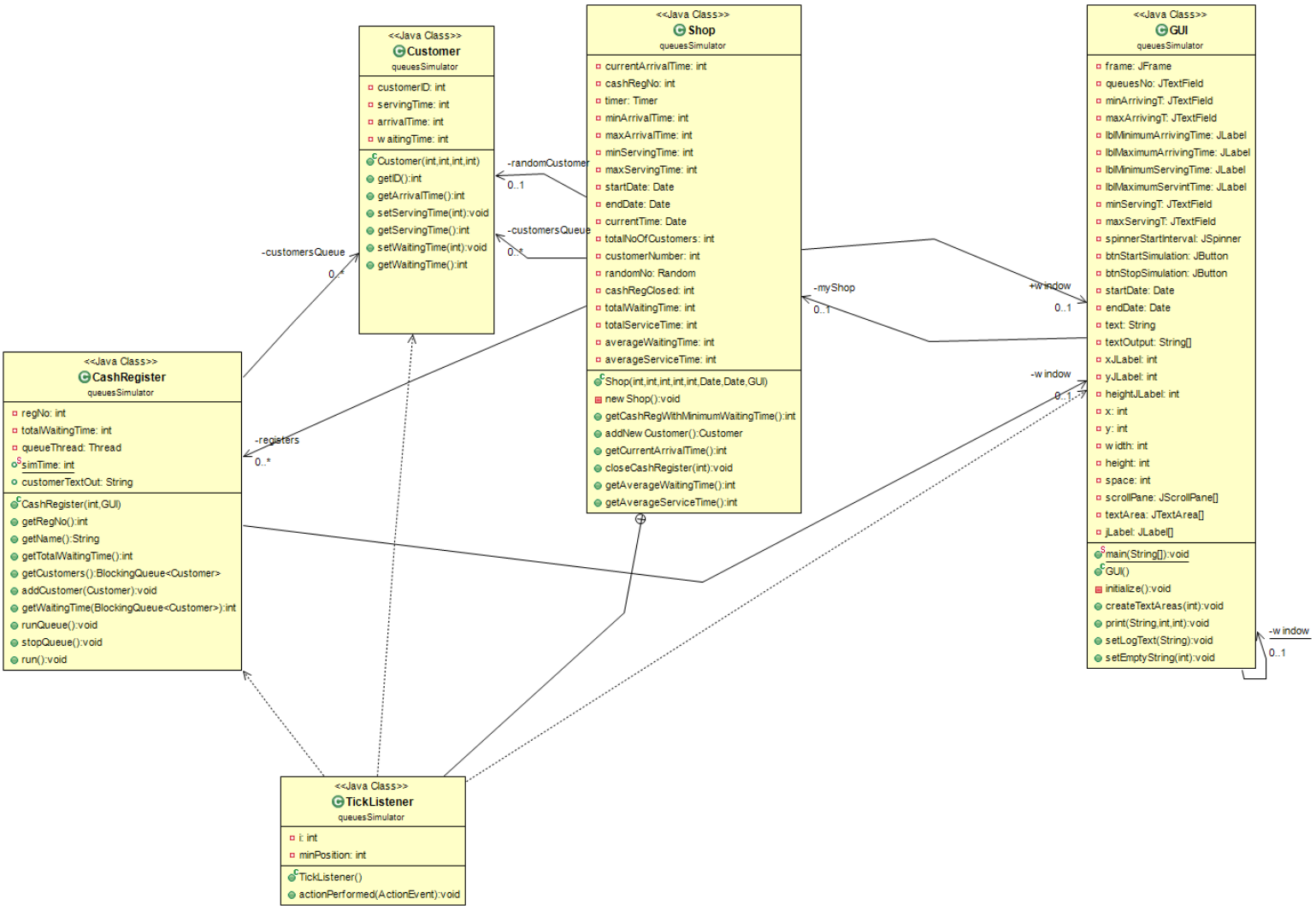
Datele de iesire in urma simularii sunt:

- Un fisier output.log in care sunt afisate toate evenimentele – sunt evidentiate sosirile tuturor clientilor, plecarea acestora, mutarea clientilor la alte cozi daca una se inchide, timpul mediu de asteptare si cel de servire.
- O evolutie in timp real a cozilor si a modului in care acestea sunt procesate, realizate sub forma unor mesaje reprezentand numarul clientilor si actiunea petrecuta: sosire, respectiv parasirea cozii.

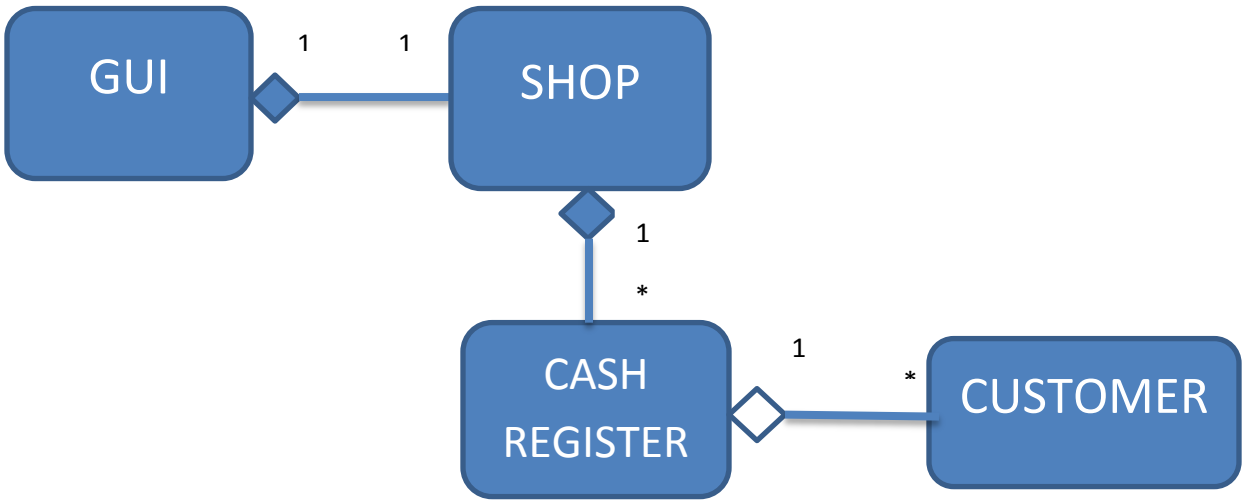
Din dorinta de a eficientiza experienta utilizatorului cu interfata si cu modul de simulare al cozilor, s-a realizat evolutia in timp real a cozilor. Mai exact, s-au utilizat campuri de text pentru a afisa numarul fiecarui client din coada, iar cand acesta pareseste coada numarul sau este sters si inlocuit de urmatorul client.

3. Proiectare

3.1 Diagrama UML



3.2 Diagrama relationala



3.3 Structuri de date

Structura de date folosita in cadrul aplicatiei pentru managementul cozilor este **LinkedBlockingQueue<E>**. Aceasta strunctura este una de tip coada, respectand principiul FIFO(first in, first out) – “primul venit, primul servit”. Intr-o astfel de structura elemental din varful cozii este cel care a fost introdus primul si a petrecut cel mai lung timp in coada, iar elemental din capatul cozii este cel mai nou introdus. Metoda de introducere a elemetelor intr-o astfel de coada este: put(), iar ce de stergere este: remove().

Ierarhia din care face parte aceasta clasa este:

- java.lang.Object
  - java.util.AbstractCollection<E>
    - java.util.AbstractQueue<E>
      - java.util.concurrent.LinkedBlockingQueue<E>

Clasa **LinkedBlockingQueue<E>** a fost aleasa datorita sigurantei cu care se lucreaza in cazul unor procesari concurente, precum cele din cazul aplicatiei, unde sunt folosite Thread-uri, acestea fiind descrise in cele ce urmeaza

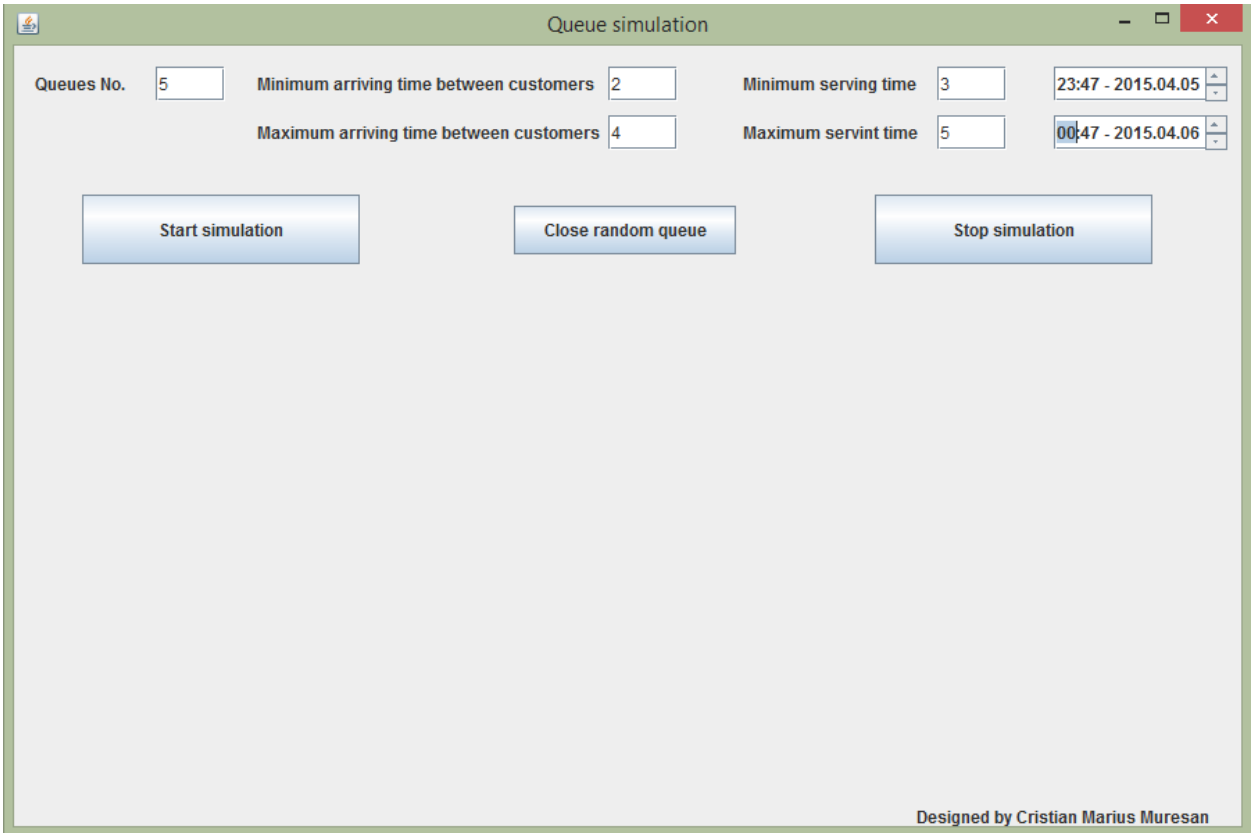
3.4 Proiectare clase

Legat de partea de proiectare, pentru rezolvarea problemei s-a ales utilizarea a 4 clase: GUI, Shop, CashRegister si Customer, aceste fiind suficiente.

1. Clasa GUI (Graphical User Interface) : **public class** GUI

Aceasta clasa, dupa cum ii spune si numele, este cea care realizeaza interfata grafica, astfel incat aplicatia sa fie usor de folosit si, in special, pentru a putea fi urmarita evolutia cozilor cu usurinta.

In cadrul clasei se gaseste metoda **public static void** main(String[] args), cea care face ca totul sa mearga, iar in urma rularii acesteia se va deschide o fereastra pe care utilizatorii o vor putea folosi sa introduca datele necesare unei simulari.



In cadrul interfetei se pot observa cinci zone de text (TextFields), trei butoate si doua elemente JSpinner.

Cele cinci zone de text reprezinta:

- Numarul de cozi ce vor fi in cadrul magazinului
- Intervalul de timp intre sosirile a doi client – acesta este generat ca o valoare aleatoare marginita de cele doua limite “Minimum arriving time between customers” si “Maximum arriving time between customers”
- Intervalul de timp rezervat serviciului – acesta este generat ca o valoare aleatoare marginita de cele doua limite “Minimum servint time” si “Maximum servint time” – fiind timpul necesar fiecarul client pentru a fi servit

Cele trei butoane reprezinta:

- Start simulation – actioneaza comanda de pornire a intregului sistem prin transmiterea tuturor parametrilor clasei Shop.
- Stop simulation – actioneaza incetarea procesarii cozilor si inchiderea intregii aplicatii. De asemenea, acest bun declanseaza scrierea in fisierul log a tuturor evenimentelor petrecute pe parcursul simularii.
- Close random queue – utilizaera acestui buton, dupa cum ii zice si numele, provoaca inchiderea unei liste si redistribuirea clientilor la listele cu cel mai mic timp de asteptare.

Cele doua elemente JSpinner:

- Sunt utilizate pentru schimbarea intervalului de rulare, acesta constand elemente precum timpul si data.

Constructorul clasei GUI - **public** GUI() – apeleaza metoda initialize();

Metodele clasei GUI:

- **private void** initialize() – in cadrul acesteia sunt intializate toate componentele din cadrul interfetei, si anume cele cinci zone de text, cele trei butoate, cele dou aelemente JSpinner si toate elementele de tip JLabel folosite pentru afisarea unor mesaje ajutatoare. De asemenea, in cadrul metodelor -**public void** actionPerformed()- utlizate la actiunea butoanelor se executa diferitele lor functii:
    - **Start simulation –** la actionarea acestui buton sunt create N zone de text pentru afisarea evolutiei cozilor, unde N este numarul de cozi, iar mai apoi se realizeaza instantierea unui obiect de tip Shop cu toti parametric introdusi in interfata : numarul de cozi, intervalul minim si cel maxim de sosire, intervalul minim si maxim de servire, limitele intervalului de simulare si, de asemenea, un obiect de tip GUI utilizat pentru apelarea metodelor de afisare ce vor fi discutate mai jos.
    - **Stop simulation –** la ationarea acestui buton se apeleaza metoda **public void** setLogText(String newText), mai apoi este creat si accesat fisierul log in

care se va tipari toata evolutia sistemului, ca mai apoi sa se apeleze System.exit(0) pentru inchiderea aplicatiei.

- **Close random queue –** la ationarea acestui buton se apeleaza metoda closeCashRegister(int) din clasa Shop, metoda ce va fi descrisa in cele ce urmeaza

- **public void** createTextAreas(**int** howMany) – aceasta metoda este utilizata pentru afisarea a celor N zone de text necesare pentru urmarirea evolutie cozilor – N este egal cu numarul de cozi. Mai precis, in cadrul metodei se creeaza N componente ScrollPane() in interiorul carora va fi cate un element TextArea().
- **public void** print(String newOutput, **int** queueNo, **int** customerID) – metoda este necesara pentru a afisa in cele N zone de text create mesaje corespunzatoare din care se intelege evolutia cozilor
- **public void** setLogText(String newText) – aceasta metoda este utilizata pentru a crea fisierul final log, care contine evolutia cozilor si lista tuturor evenimentelor din sistem.
- **public void** setEmptyString(**int** i) – metoda este folosita pentru stergerea continutului unei zone de text la inchiderea cozii associate.

2. Clasa Shop : public class Shop

Clasa Shop este clasa care coordoneaza intreaga simulare, de la introducerea clientilor, pana la inchiderea unei cozi.

Constructorul clasei Shop - **public** Shop(**int** cashRegNo, **int** minArrivalTime, **int** maxArrivalTime **int** minServingTime, **int** maxServingTime, Date startDate,Date endDate, GUI window) – preia toti parametrii necesari simularii din interfata, si ofera accesul la acestia pentru a efectua simularea. De asemenea, acesta apeleaza metoda **private void** newShop(), care va fi descrisa curand, iar tot aici este creat un timer cu intarziere de o secunda – 1000 de milisecunde - ce creeaza un obiect de tip TickListener(), aceasta fiind o clasa interna clasei Shop.

Clasa interna TickListener : **public class** TickListener **implements** ActionListener – aceasta clasa interna este apelata in functie de timer, iar de aici sunt sincronizate toate actiunile urmatoare. Pentru aceasta, clasa implementeaza interfata ActionListener. Singura metoda din cadrul clasei, **public void** actionPerformed(ActionEvent arg0), este accesata de o data pe secunda, dupa cum este setat timer-ul. Aici, in interiorul metodei, este generat random un interval de sosire cuprins intre minim si maxim, se introduce client noi lasa registrul cu cel mai mic timp de asteptare si mia apoi se apeleaza afisarea unui mesaj corespunzator pentru a se obserbva evolutia cozii in interfata. Acest timer este oprit in momentul in care timpul current de rulare este egal sau mai mare decat limita superioara intervalului de simulare.

Metodele clasei Shop:

- **private void** newShop() – in aceasta metoda sunt deschise (create) cozile (obiecte de tip CashRegister), iar dupa deschidere este apelata metoda runQueue() din clasa CashRegister pentru a porni Thread-urile care proceseaza cozile. Tot in metoda newShop() sunt setate la valoarea 0 secundele si milisecundele timpilor de inceput si sfarsit ai simularii, pentru a se putea face comparatia intre cele doua fara problem. In functie de interval, este apelata sau nu metoda start a timer-ului creat precendent pentru sincrinizarea intregii aplicatii.
- **public int** getCashRegWithMinimumWaitingTime() – aceasta metoda returneaza numarul cozii cu cel mai mic timp de asteptare, coada la care va fi introdus urmatorul client.
- **public** Customer addNewCustomer() – metoda addNewCustomer() este utilizata pentru a adauga un client nou unei cozi in functie de parametri introdusi in interfata. Aceasta returneaza un obiect de tip Customer.
- **public void** closeCashRegister(**int** i) – aceasta metoda realizeaza inchiderea cozii cu numarul I ,generat aleator, mutarea clientilor de la acea coada spre alta cu timpul de asteptare cel mai mic si afisarea in interfara si in fisierul log a evenimentelor petrecute.
- **public int** getAverageWaitingTime() – returneaza timpul mediu de asteptare al magazinului, in functie de timpul total de astepare al cozilor impartit la numarul de client care au vizitat magazinul
- **public int** getAverageServiceTime() – returneaza timpul mediu de servire in functie de timpul total de servire al fiecarui client si impartit la numarul de client.

3. Clasa CashRegister: **public class** CashRegister **implements** Runnable

Clasa CashRegister implementeaza interfata Runnable, necesara pentru rularea Thread-urilor, astfel incat fiecare coada sa fie procesata independent de celelalte. Interfara Runnable va fi descrisa mai pe larg intr-un subcapitol urmator.

Un obiect de tipul CashRegister contine un numar (un ID), un obiect de tip GUI pentru a putea realiza tiparirea in cadrul interfetei si in fisierul log, si, de asemenea, o structura de tipul LinkedBlockingQueue<Customer> in care este memorata coada ce urmeaza a fi procesata.

Constructorul clasei CashRegister: **public** CashRegister(**int** regNo, GUI window) – initializeaza numarul registrului si obiectul de tip GUI, si totodata structura de coada folosita.

Metodele clasei CashRegiste:

- **public int** getRegNo() – metoda returneaza numarul registrului
- **public** BlockingQueue<Customer> getCustomers() – metoda returneaza un obiect de tip BlockinQueu<Customer>
- **public void** addCustomer(Customer newCustomer) - aceasta metoda realizeaza introducerea unui nou client in structura de tip coada. Introducerea se face prin apelul metodei put(Object obj), iar acesta se adauga la capatul cozii.
- **public int** getWaitingTime(BlockingQueue<Customer> customersQueue2) - metoda returneaza timpul de asteptare al unei cozi intr-un anumit moment al rularii, cand este apelata
- **public void** runQueue() – aceasta metoda este utilizate pentru a porni fiecare thread in parte, prin apelul start() efectuat asupra unui thread.
- **public void** stopQueue() – efectueaza opusul metodei precedente, adica dezactiveaza un thread cand este actionat butonul Close Random Queue din interfata.
- **public void** run() – aceasta metoda este, de fapt, metoda suptrasrcisa din interfata Runnable. Aceasta se apeleaza dupa pornirea fiecarui thread, iar aici se verifica tot timpul daca coada asociata unui registru este goala, caz in care se continua veficiarea, iar in caz contrar thread-ul este adormit pentru un interval egal cu timpul de servire al fiecarui client, iar mai apoi acesta urmeaza a fi scos din coada.

4. Clasa Customer: public class Customer

Clasa Customer reprezinta elemental de baza al acestei simulari, deoarce fara client nu ar fi nimic ramas de simulat.

Constructorul clasei Customer: **public** Customer(**int** customerNumber, **int** arrivalTime, **int** servingTime, **int** waitingTime) – unde customerNumber este ID-ul fiecarui client, arrivalTime este timpul de sosire al fiecarui client, servingTime este timpul necesar serviciului asteptat de fiecare client, iar waitingTime este initial 0, insa acesta va fi setat mai tarziu.

Metodele clasei Customer:

- **public int** getID() – returneaza ID-ul fiecarui client
- **public int** getArrivalTime() – returneaza timpul sosirii fiecarui client
- **public void** setServingTime(**int** newServingTime) – seteaza timpul de asteptare afferent fiecarui client
- **public int** getServingTime() – returneaza timpul de servire
- **public void** setWaitingTime(**int** newWaitingTime) – seteaza timpul de asteptare in urma unor modificari

- **public int** getWaitingTime() – returneaza timpul de asteptare al unui client in coada sa.

3.5 Pachete si interfete

Un pachet Java este un mecanism de organizare a claselor Java. Pachetele Java pot fi stocate in fisiere comprimate numite fisiere JAR, permitand claselor sa fie accesate mai repede ca si grupt decat cate una pe rand. Programatorii folosesc, de asemenea, pachete pentru a organiza clasele ce apartin aceleiasi categorii sau care asigura o functionalitate asemanatoare. Un pachet asigura un spatiu cu nume unic pentru tipurile pe care le contine. Clasele din acelasi pachet pot accesa membri care au acces ai celeilalte clase. Un pachet permite unui dezolvator sa grupeze clasele si intefetele.Aceste clase trebuie sa aiba o legatura.

Pentru acest proiect, au fost importate urmatoarele pachete:

- import java.awt.event
    1. **import** java.awt.event.ActionEvent;
    2. **import** java.awt.event.ActionListener;
- import javax.swing
- import java.util.concurrent.BlockingQueue;
- import java.util.concurrent.LinkedBlockingQueue;
- import java.util.logging.*;

3.5.1 Interfata Runnable

Una dintre cele mai foloste inferfete in Java este interfata Runnable. Este foarte importanta deoarce este implementata de clasa Thread.

Interfata Runnable asigura un protocol comun obiectelor care doresc sa-si execute codul atata timp cat sunt active.Un obiect este activ daca a fost lansat in executie si nu a fost oprit. Inima unui obiect, care implementeaza interfata Runnable este metoda run(). Aceasta metoda trebuie suprascrisa. De altfel, este singura metoda a interfetei Runnable.

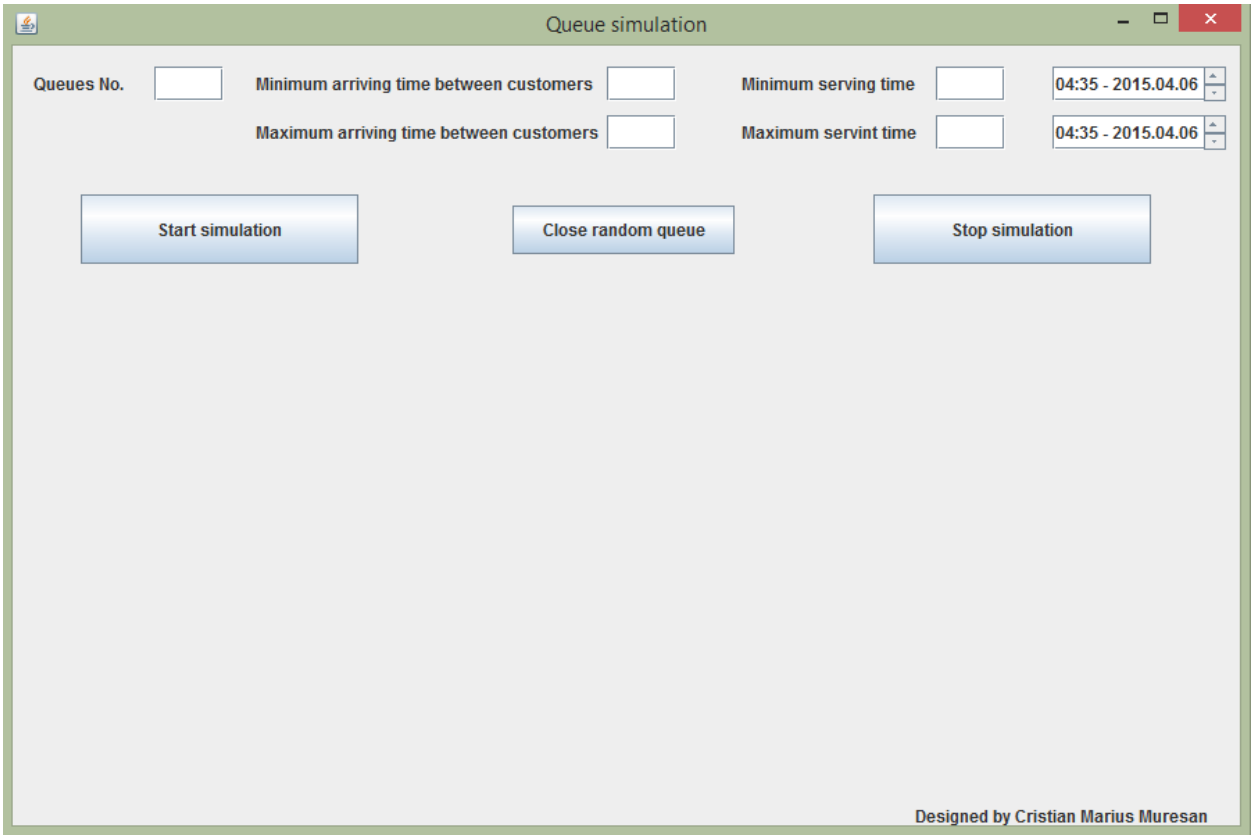
Interfata se numeste Runnable si nu Running, deoarece ea nu se executa chiar tot timpul. Pe marea majoritate a masinilor se afla un singur processor care e ocupat si cu alte sarcini. Doar o parte din timpul de lucru al procesorului este alocata obiectulului de tip Runnable.

In acest program avem un array de obiecte de tip CashRegister in clasa Shop, iar pentru fiecare se creeaza un thread now, astfel incat executia lor sa fie concurenta. Un exemplu de implementare a unui nou Thread folosind interfata Runnable este:

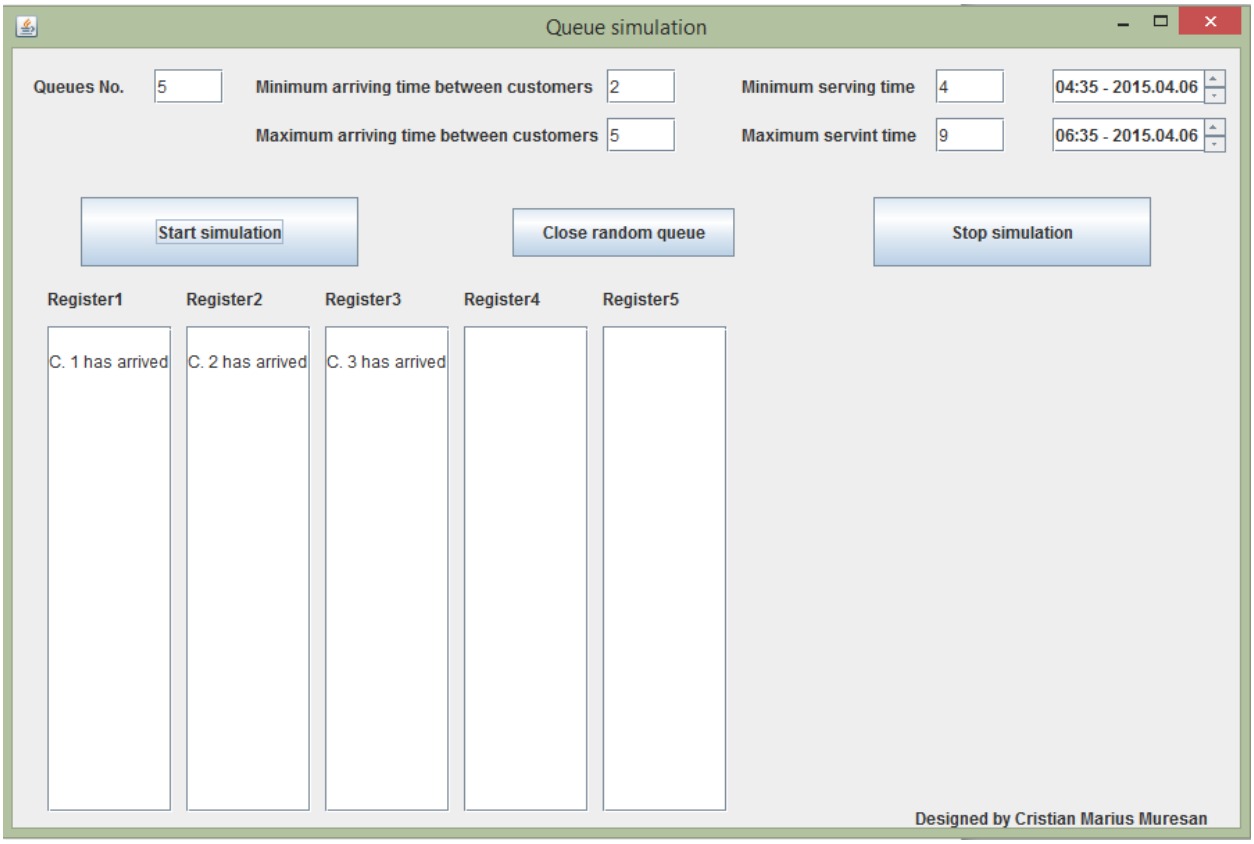
```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }
}
```

3.6 Interfata utilizator

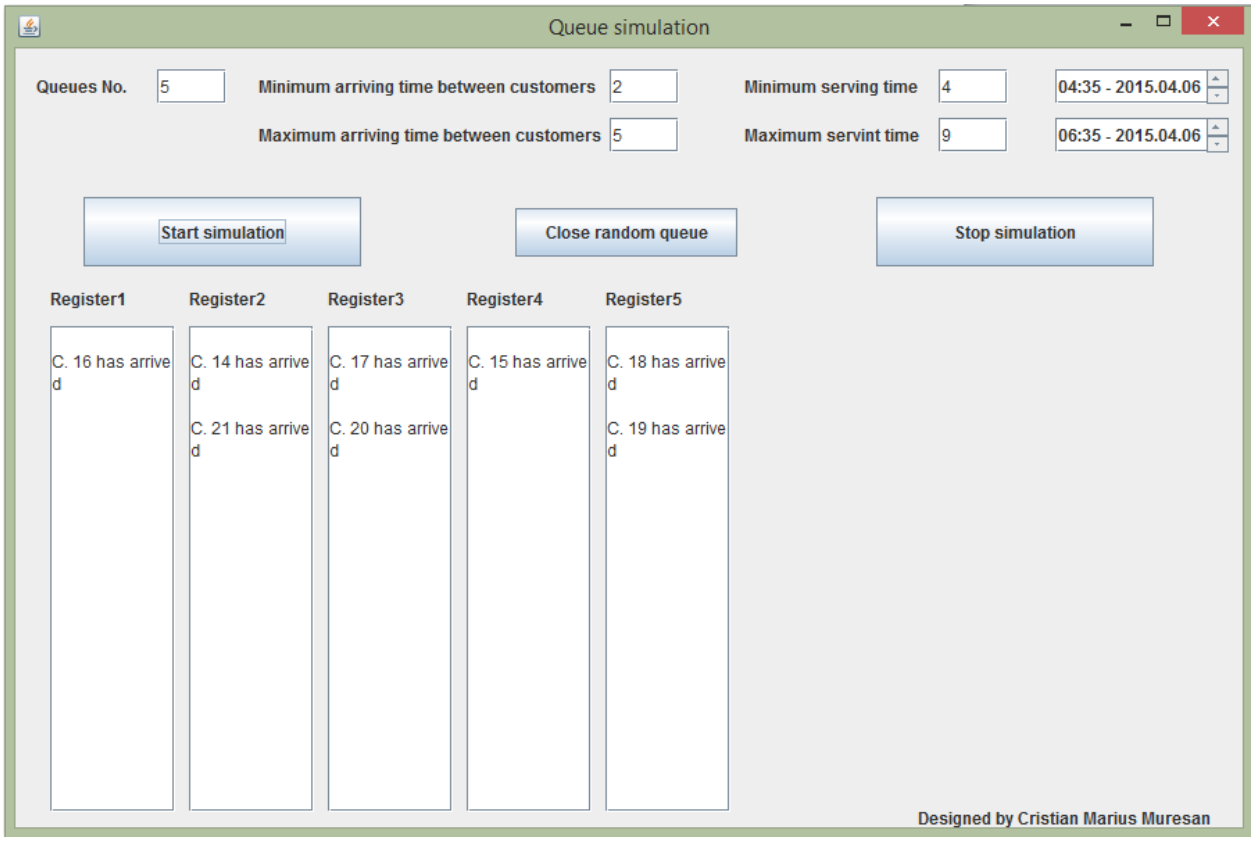
Legatura dintre utilizator si aplicatia realizata este constituita din interfata grafica, aceasta fiind partea la care are acces orice utilizator. Aceasta este formata din cinci campuri de text unde se introduce datele necesare unei simulari, trei butoane care declanseaza diferite actiuni si doua componente de tip JSpinner.

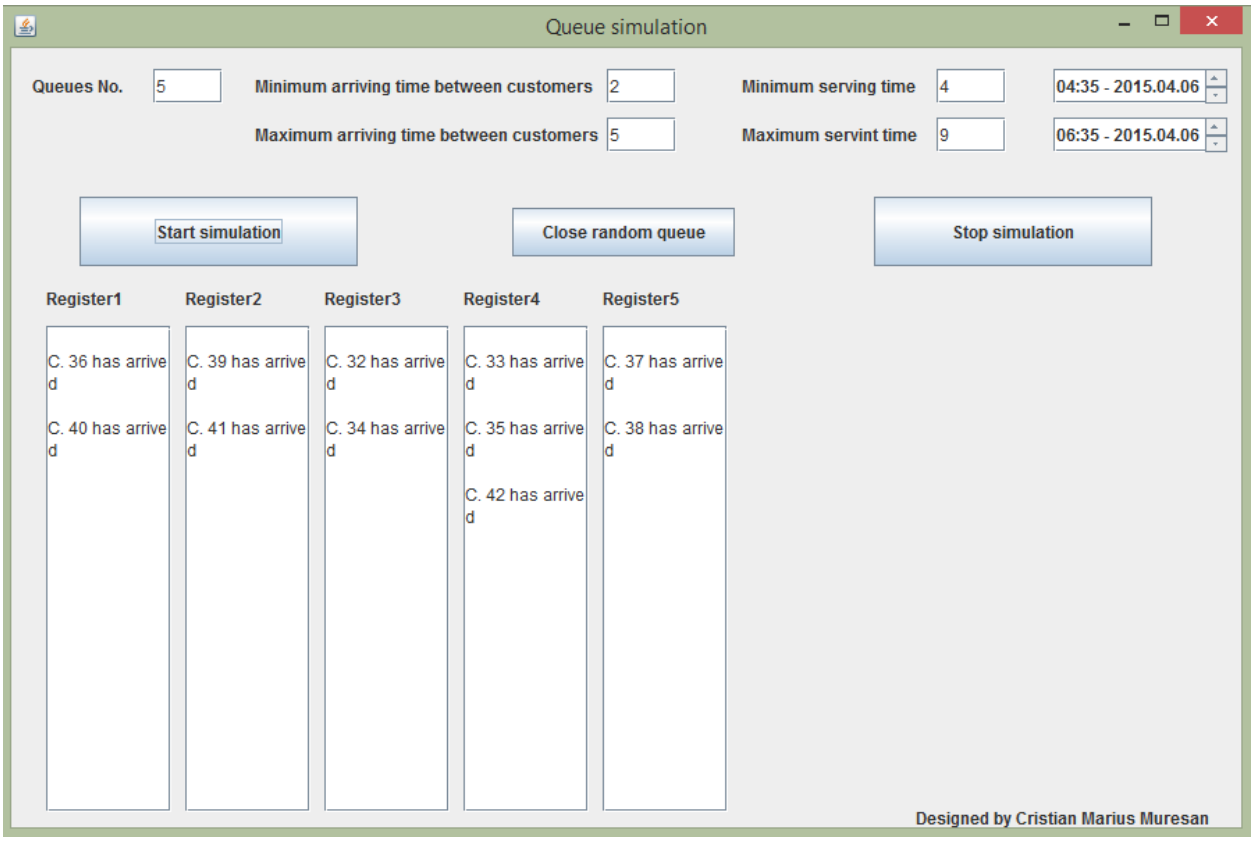


De asemenea, in timpul executiei, in zona inferioara a ferestrei vor aparea atatea campuri de text, cat este numarul registrelor introdus in TextField-ul Queues No. Obtinerea numarului de register se face prin: Integer.parseInt (textFieldQueuesNo.getText()), dupa cum urmneaza:



Evolutia cozii se poate verifica in urmatoarele doua poze:





La finalizarea simuarii, evenimentele petrecute pe tot parcursul acesteia vor fi afisate intr-un fisier log, exemplul dat reprezentand sfarsitul listei:

```
"Customer number 42 has arrived at Mon Apr 06 06:33:00 EEST 2015 at
register: 4
Customer number 30 has left the store.
Customer number 37 has left the store.
Customer number 33 has left the store.
Customer number 32 has left the store.
Customer number 36 has left the store.
Customer number 38 has left the store.
Customer number 39 has left the store.
Customer number 35 has left the store.
Customer number 34 has left the store.
Customer number 40 has left the store.
Customer number 42 has left the store.
Customer number 41 has left the store.
The average waiting time for this store is: 12 minutes.
The average service time for this store is: 5 minutes."
```

Dupa cum se poate observa, sistemul monitorizare a cozii functioneaza bine, iar pentru simularea efectuata timpul mediu de asteptare este de 12 minute, iar cel de asteptare este de 12.

4. Implementare si testare

Partea de testare se va face folosind fisierul QueuesSimulator.jar, utilizarea aplicatiei facandu-se precum in prezentarile anterioare. Utilizatorul este invitat a introduce date aleatoare in sistem si a genera diferite secvente de simulare.

5. Rezultate

Spre exemplu, pentru un magazine cu 5 cozi, timpul de sosire a unui nou client intre 2 si 4 si timpul de servire intre 3 si 6, rezultatele sunt:

-Timpul mediu de asteptare la coada este: 7 minute.

-Timpul mediu de servire a unui client este: 3 minute.

6. Concluzii

La prima lectura a cerintei problemei, realizarea acestei aplicatii nu parea a fi complicata, gandindu-ma la cunostintele matematice necesare implementarii unor cozi, insa, la o analiza mai complexa se poate deduce ca pot aparea detalii neprevazute, cum ar fi probleme de implementare iar alegerea unor structuri de date care sa lucreze in bune conditii cu thread-uri si cu cozi.

Prin aceasta tema, in primul rand, am invatat despre lucrul cu Thread-uri. Ce sunt. Cum se folosesc. La ce ajuta. Nu a fost tocmai usor la inceput, dar nici nu trebuia sa fie.

In al doilea rand, am reusit sa inteleg mai bine conceptele de OOP si sa vad solutia de ansamblu in cazul unei problem de acest gen.

Dezboltari ulterioare:

- o interfata grafica cu animatie pentru a afisa précis miscaririle tuturor clientilor

- implementarea unei optiuni de opri simularea si de a o reporni

7. Bibliografie

[1] https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html

[2] https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/LinkedBlockingQueue.html

[3] https://docs.oracle.com/javase/tutorial/uiswing/misc/timer.html

[4] Head First Java, 2nd Edition

[5] http://users.utcluj.ro/~jim/OOPE/lectures.html