Shop Simulation

Student:Mocan Ioana

Grupa:30223

Cuprins

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**

Obiectivul temei prezentate este acela de a simula aparitia la o coada a unor clienti intr-un magazin.Simularea consta in aparitia la diferite momente de timp in functie de timpul de sosire si timpul de servire. Fiecare client are determinat aleator un timp de servire si un timp si u timp de sosire.

Fiecare valoare si de sosire si de servire este determinat de un min si un maxim random. Lungimea este determinata in functie de cati clienti se afla pana in prezent la coada si timpul de servire maxim.

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

Prin analiza problemei ne dam seama ca avem nevoie de concepte care privesc lucrul cu programare obiectuala.Problema a fost analizata din punct de vedere al unui magazin care are mai multe case de marcat si alegerea alatoare a caselor dar in functie de timpul minim la de servire la fiecare casa. Problema este una care permite diferite abordari si nu este exclusa ideea ca fiecare abordare sa aiba rezultate diferite.In cazul dezvoltari problemei elementele incluse in rezolvarea problemei variaza de la ca la ca ele sunt fie mai bine conturate in ceea ce priveste incapsularea, mostenirea, polimorfismul, abstractizarea.Pentru a putea implementa corect problema este necesar sa fim atenti la datele necesare si la comunicarea cu exteriorul.

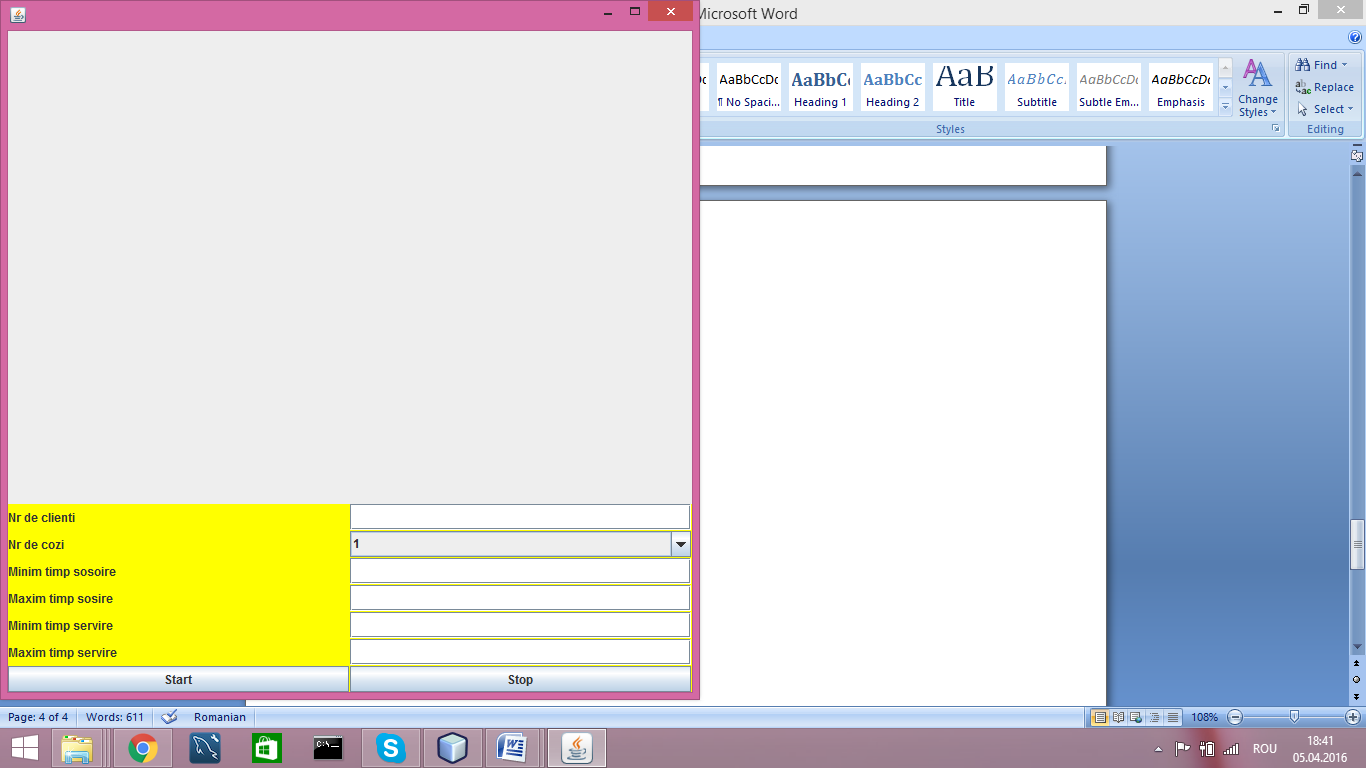
In analiza problemei a luat in considerare faptul ca avem un magazin cu mai multe case care se pot inchide in functie de eliberare daca mai multe case se inchid va ramane doar casele cu clientii prezenti.La fiecare casa trebuie sa se ia in considerare timpul minim si se calculeaza unde este minim si acolo se va pune urmatorul client.Un client apare la coada in momentul in care termina cumparaturile din magazin.Pentru simulare se alege numarul de clienti care se afla la magazin cate case sunt in magazin si se ma alege si un minim si un maxim intre care se alege random timpul de sosire si servire.Singura operatie prezenta aici este aceea de simulare a mai multor cozi de diferite lungimi.

Un aspect care conteaza in relatia utilizator-produs este crearea unei interfete cat mai prietenoase si care sa scoata in evidenta cat mai bine utilizarea codului.

Din punct de vedere a implementari aplicatiei exista mai multe optiuni de implementare ceea pe care am ales-o eu este una care poate fi imbunatatita. Varianta pe care am ales-o sa o implementez poate sa lase putin de dorit deoarece nu simuleaza varianta reala a unui magazin cu ar fi sa alegi aleator casa si sa astepti un timp indelungat pana esti servit sau sa stea cat mai mult in magazin. Am incercat sa evidentiez cum functioneaza alegerea in functie de timpul minim a unei case.

Utilizarea aplicatiei este descrisa in ceea ce urmeaza:

* Informatia de care avem nevoie este luata de din textField-uri si din comboBox
* Odata completate textField-uri si din comboBox se apasa pe butonul de start care genereaza aleator toate timpul de sosire si timpul de servire pentru fiecare iar mai apoi se genereaza timpul mediu de servire.
* Informatia generata este in functie de valorile care sunt transmise prin min si max timpului de servire si sosire.
* Timpul mediu este generat atata timp cat aplicatia inca functioneaza deci pentru fiecare client ramas se genereaza alta valoare cu cat nr clientilor scade cu atat se genereaza mai departe valori mai mari de timp.



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

Penru implementarea aplicatiei este necesara utilizarea a 7 clase, fiecare fiind specializata in solutionarea unor operatii specifice.

**public class Shop**

Aceasta clasa are rolul de a apela clasa View care creeaza interfata si utilizarea simulatorului de cozi la n case .Practic are rolul de a rula aplicatia si preluand tema sistemului de operare pentru a crea casetele cu interfata grafica si pentru a o expune pentru utilizator.

**public class Client**

In aceasta clasa sunt constructorul si functiile de get si set care preiau id-ul ,timpul de sosire , timpul de servire . Mai avem metoda de verificarea daca o persoana se afla la casa sau nu si metoda prin care cream un panel care reprezinta clientul.

**public class Coada extends Thread**

In aceasta clasa sunt constructorul care initializeaza lista de clienti aici se afla metoda de get si set pentru lista de clienti.Pe langa avem si functie de adugare unui client in lista.Mai avem metoda de determinare a timpului cel mai scurt pentru o clasa. In aceasta clasa ii prima data cand folosim Threadul aici cream un thread care reprezinta o coada creata din diferiti clienti cu diferite timpuri de sosire si servire.In aceasta clasa avem un constructor care initializeaza lista de clienti coada in care se aseaza clienti si Frame-ul in care urmeaza se scrie. Aici pentru prima data suprascriem metoda **run()** care ne ajuta sa rulam acest thread.

**public class Server extends Thread**

In acesta clasa este a doua oara cand folosim Thread ca sa cream thread-uri care reprezinta fiecare cate o coada numarul de threduri este determinat de numarul dat de la tastatura care reprezinta numarul de case.In acesata clasa avem un constructor care initializeaza lista de clienti lista de cozi si timpul de rulare si frameul in care se va afisa mai apoi cozile.In aceasta clasa suprascriem din nou metoda run() care ne ajuta sa rulam thread-urile.In aceasta clasa am intodus o metoda de sortarea care consta in verificarea daca timpul de sosire al unui client este mai mare decat al celui de-al doilea si se mai verifica si daca timpul de servire este mai mare decat la cel de-al doilea daca conditiile sunt indeplinite se aplica o interschimbarea .Aceasta metoda returneaza o lista de clienti.

**public class View implements ActionListener**

Clasa View are ca rol implementare a interfetei grafice si asigurarea ciclului de informatie dinspre interior sper exterior si invers.Toata implementarea grafica se realizeaza cu ajutorul unui panel care este raspunzator de datele de intrare ,de butoane.

Interfata grafica este realizata din 2 butoane care se ocupa de start si stop.Pe langa butoane se mai gasesc si label-uri care ajuta la intalegerea corecta a semnificatiei datelor introduse.

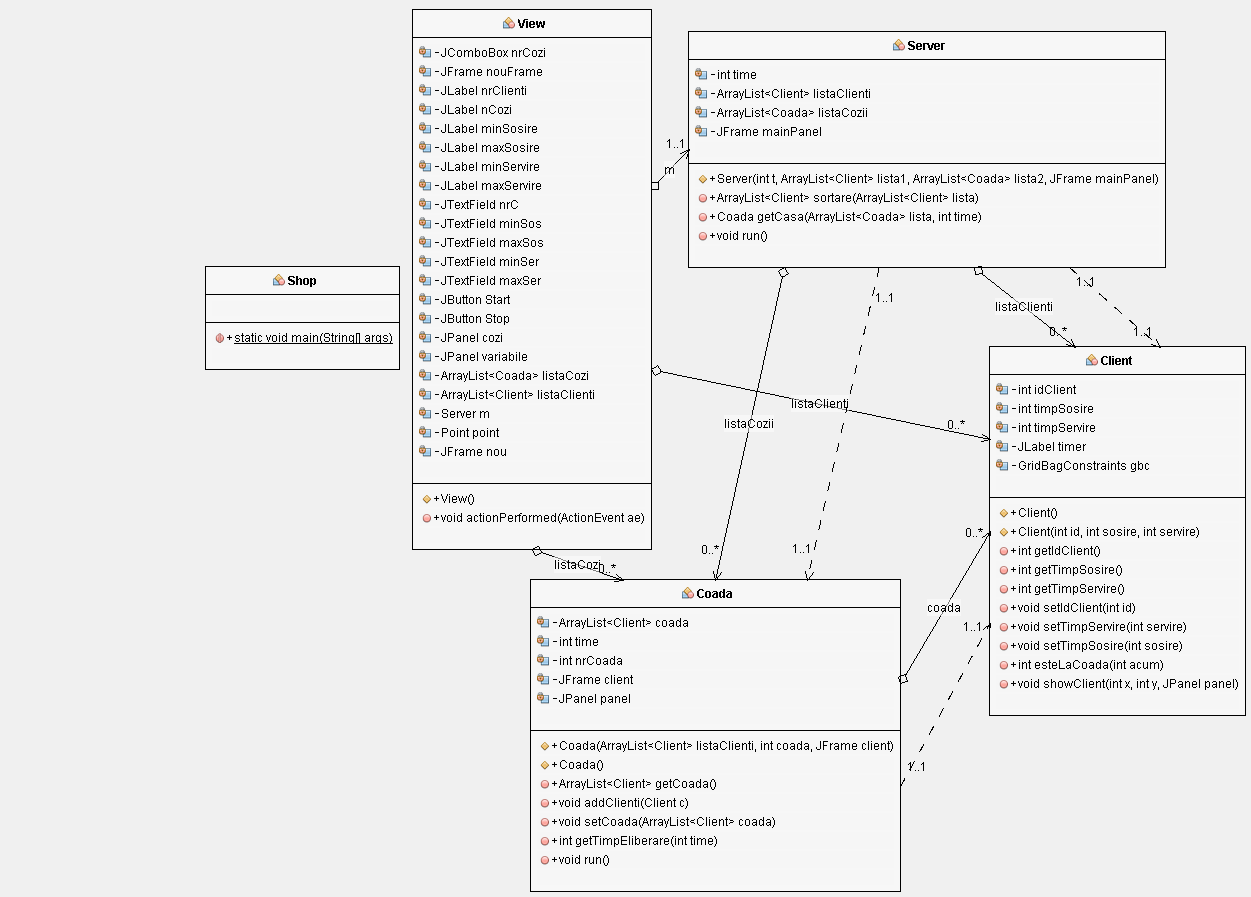
Fiecare buton este responsabil sa indeplineasca o operatie singulara si care sa faca ceea ce ii este cerut.

Butonul „Start” are rolul de a incepe rularea threadurilor si de a extrage informatia din textFieldurile si din comboBox pentru a ajuta la simularea unui magazin.

Butonul „Stop” opreste rularea thredurilor si se opreste timpul de rulare mediu care este determinat de timpul total supra numarul de clienti .

**4. Implementare si testare**

Dupa cum se observa din diagrama UML s-au respectat paradigmele programarii orientate pe obiect. Abstractizarea se poate observa prin evidentierea elementele indispensabile.Conceptul de incapsulare se verifica la declararea private a elementelor din fiecare clasa , acestea putand fi accesate doar cu ajutorul unor metode de tip get si set.



In ceea ce urmeaza o sa explic metodele mai importante din fiecare clasa:

**public class Client**

Una dintre metodele importatnte din interiorul clase Client este aceea **public int esteLaCoada(int acum)**  in care se verifica daca un client se afla la coada sau nu inca in functie de timpul de sosoire si cel de servire. Se verifica daca timpul de sosire este mai mic decat timpul actual si daca suma dintre timpul de sosire si timpul de servire este mai mare sau egal cu momentul actual iar daca aceste doua conditii sunt indeplinite atunci clientul este in coada.

A doua metoda importanta din clasa client este public **void showClient(int x,int y,JPanel panel)**  este aceea de afisare a panelului in care se afla clientul se afiseaza la anumite coordonate si de anumite dimensiuni.

**public class Coada**

**public int getTimpEliberare(int time)** in aceasta metoda verific daca daca un anumit client se afla la coada la timpul respectiv si daca se afla cresc timpul de asteptare cu 2 pentru fiecare client din lista .

**public void addClienti(Client c)**  in aceasta metoda adaug in coada cate un client.

In acesata clasa suprascriem metoda **run()**. In aceasta metoda **public void run()** pe care am suprascris-o facem urmatoare operatii cream coada intr-un nou panel. In aceasta metoda preiau timpul de sosire al primului client si mai apoi adaug la acesta timpul de servire al fiecarui client cu acest timp deja determinat parcurg cu ajutorului unui for toate valorile pana la valoarea de timp determinata si mai apoi iau fiecare client din lista si verific daca clientul se afla la coada in momentul actual care este o valoare intre 0 si valoarea determinata daca este 1 se determina nivelul la care se afla si mai apoi daca este diferit de 0 se seteaza timpul de servire de la un client cu valoarea timpului de servire a clientului respectiv plus 2.Mai apoi se creeaza panelul si coada.

Cea mai simpla metoda de a crea un fir de executie care sa realizeze o anumita actiune este prin extinderea clasei Thread si supradefinirea metodei run a acesteia.Si astfel se utilizeaza aceasta clasa deoarece in acest fel , orice clasa ce descrie fire de executie va contine metoda run in care este implementat codul ce va fi rulat.

**public class Server extends Thread**

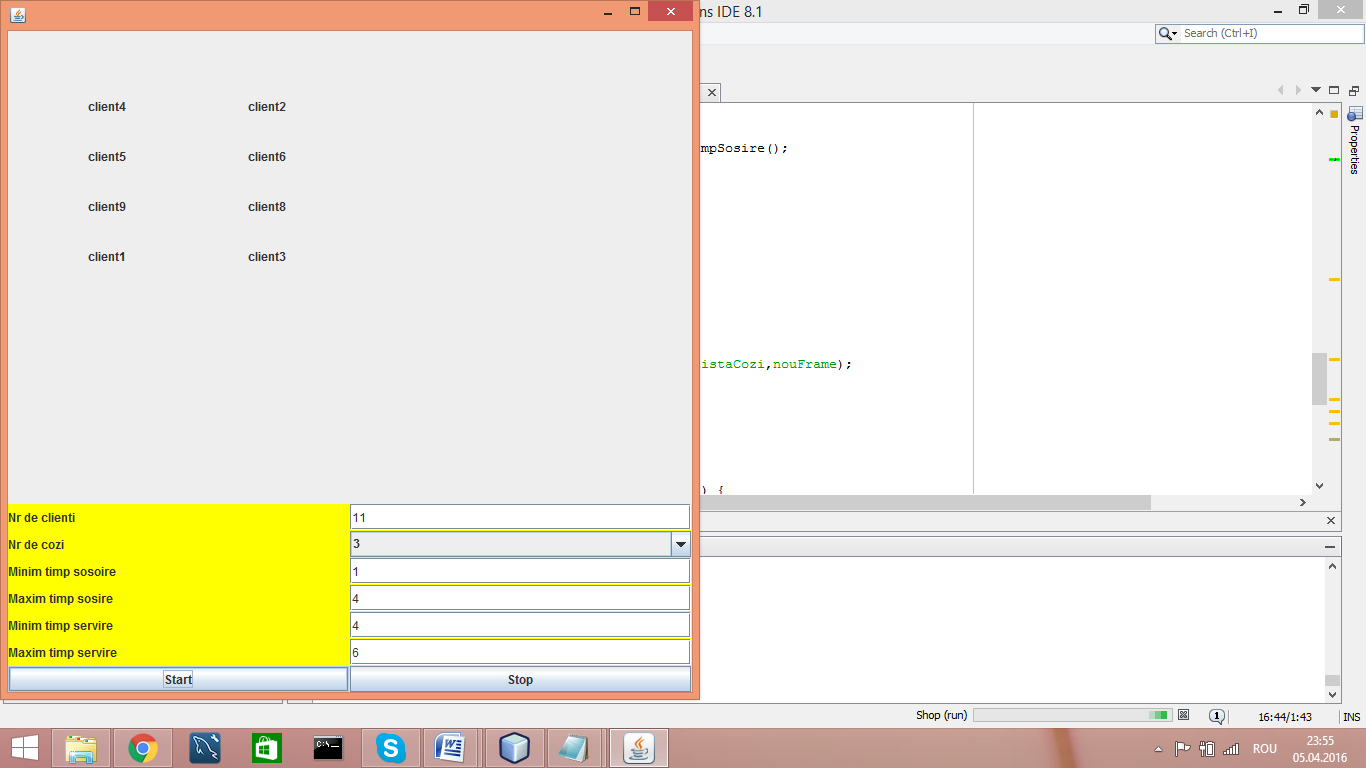
In acesata clasa suprascriem metoda run().In aceasta metoda **public void run()**  pe care am suprascris-o facem urmatoarele operatii cream n cozi intr-un nou panel care mai apoi se va afisa in interiorul Frame-ului.Se creaza o metoda care sorteaza lista de clienti in functie de timpul de sosire si timpul de servire.Parcurgem lista de clienti in interiorul for-ului de parcurgere a clientilor parcurgem cu ajutorului altui for o valoare de timp de la 0 pana la timp si daca timpul de sosire a unui este egal cu indexul care reprezinta timpul atunci alegem o casa cu ajutorul unei alte metode in care sa introducem un client.Mai apoi parcurg lista de cozi si creez cate un thread pentru fiecare coada.La sfarsit determin timpul mediu de servire pentru diferite valori ale timpului actual.Metoda pe care am mentionat-o mai sus cea care alege casa la care sa fie asezat un anumit client este de forma **public Coada getCasa(ArrayList<Coada> lista,int time)**  in aceasta metoda verific daca o anumita coada este vida se returneaza coada respectiva daca nu este se determina timpul minim din fiecare coada si cel cu timpul de eliberare minim si se atribuie unei variabile care retine coada minima valoarea la care s-a ajuns pana in momentul de fata adica la ce coada se afla minimul , mai apoi se returneaza lista care are indicele de coada minima.

La ambele clase care implementeaza Thread-uri au doar un singur constructor care in principiu atribui valori elementelor specifice thread-urilor scrise, evident, se pot defini si alti constructori, acestia fiind utili atunci cand vrem sa trimitem diversi parametri de initializare a firului nostru. A doua metoda este metoda run, ”inima” oricarui fir de executie, in care scriem efectiv codul care trebuie sa se execute. Un fir de executie creat nu este automat pornit, lansarea sa fiind realizeaza de metoda start, definita in clasa Thread.

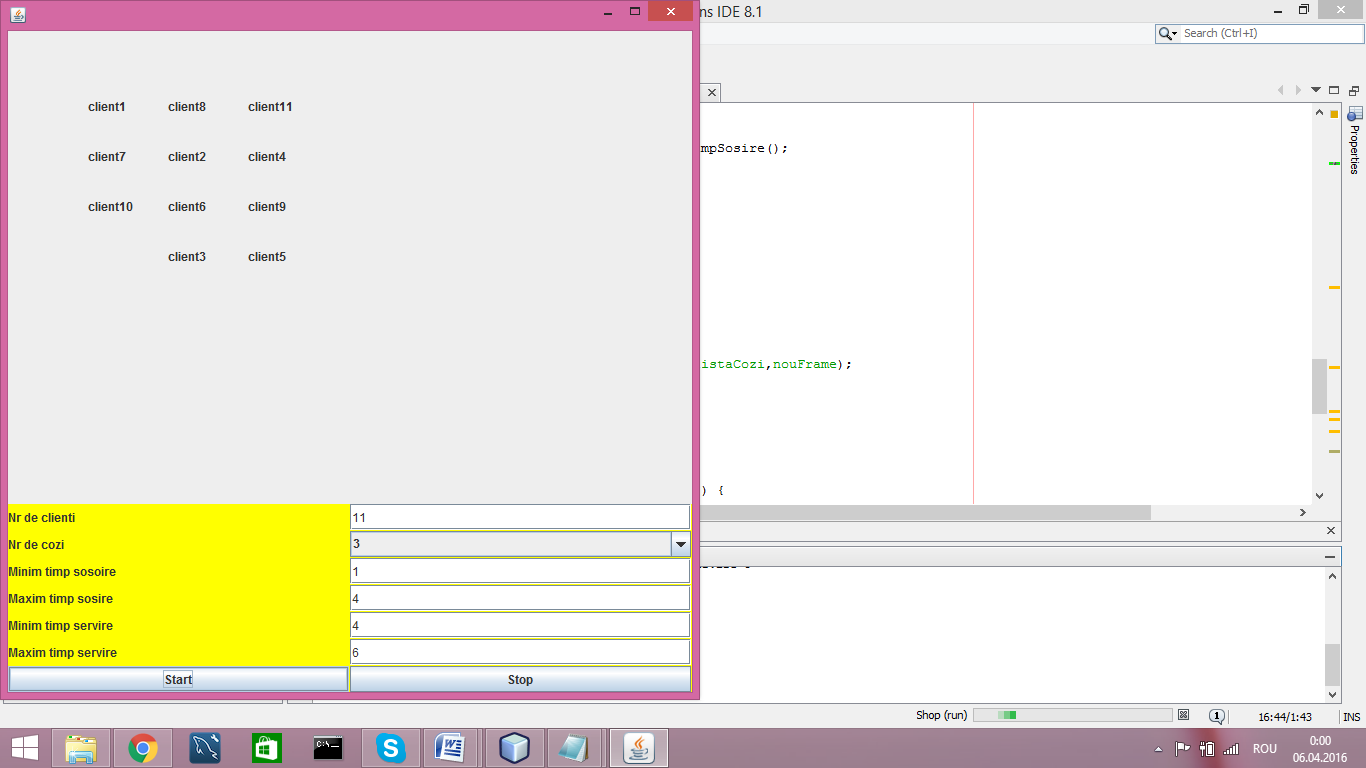
In clasa **public class View implements ActionListener**  aici se aplica un actionListener pe butonul de Start care preia informatia de care avem nevoie sa ne cream simularea cum ar fi numarul de clienti care se afla in magazin numarul de case care se afla in magazin, timpul minim de sosire si timpul maxim se genereaza aleator mai multe valori in interiorul acestui interval pentru fiecare client si apoi se preia timpul minim si maxim de servire care si acesta se determina random pentru fiecare client.Si apoi se determina timpul de rulare . Cu ajutorul acestor valori se creeaza mai apoi thread-urile adica cate case se afla in magazin.

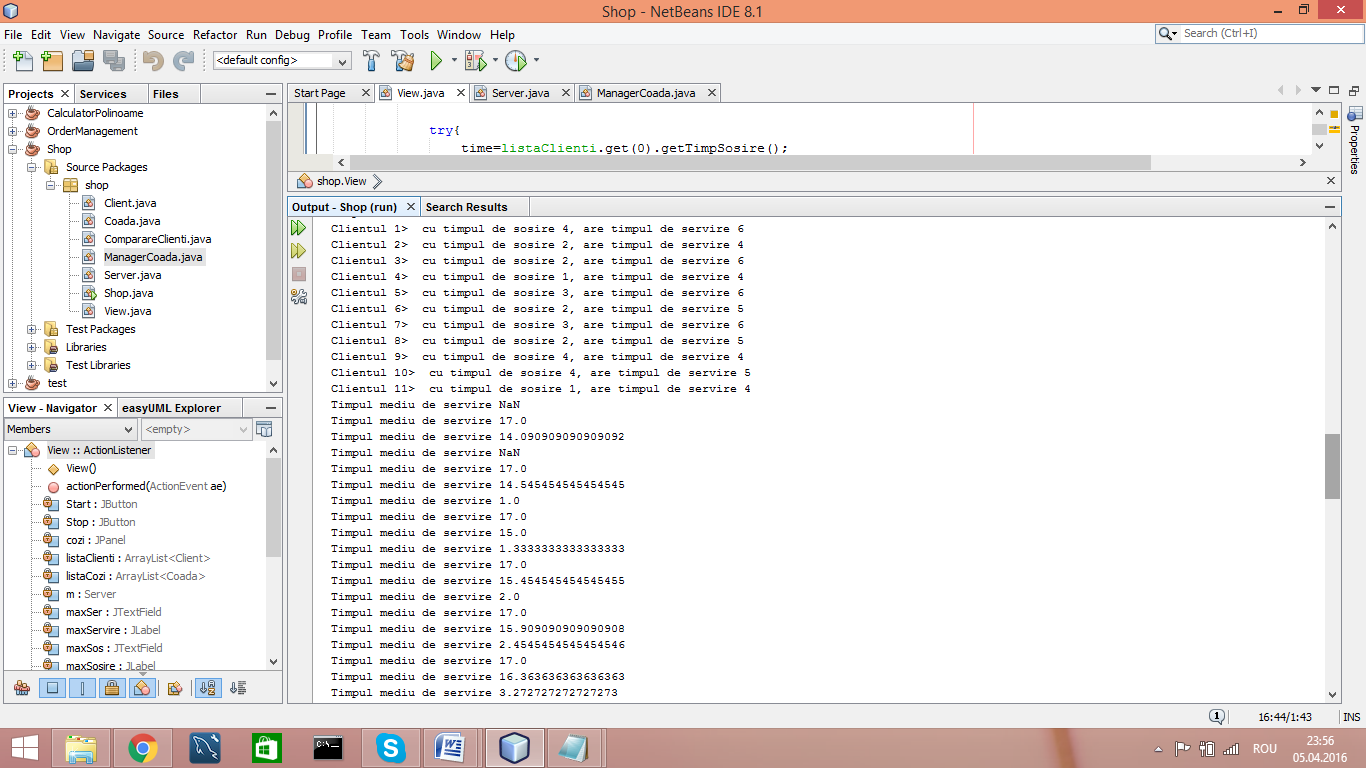
**5. Rezultate**

In acesata etapa se va testa functionalitatea aplicatiei pentru functiile pe care acestea trebuie sa le implementeze verificand astfel si corectitudinea algoritmilor implemnentati.



Dupa cum se vede s-au pus clienti la doua dintre case dar din cauza ca clientii au fost serviti pana ce la cealata casa se mai pune nu am reusit sa surprind o astfel de simulare.

Aici se observa cum la fiecare din cele 3 clase se afla clientii aceasta aranjare in functie de timpul random generat pentru fiecare client.Se poate observa ca pentru acelasi valori cozile de clienti pot sa se completeze aproape in acelasi timp dar si in momente diferite.



Aici se observa timpul de sosire si timpul de servire pentru fiecare client si mai apoi timpul mediu de servire in functie de timpul de rulare al aplicatiei.

**6. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare**

Prin aceasta tema s-au evaluat conceptele de baza ale ale programarii orientate pe obiect concepte care m-au ajutat sa imi insusesc mai bine unele notiuni legate pe programarea orientata pe obiect.

Dezvoltari ulterioare:

* Dezvoltarea unei interfete mai atractive care sa evidentieze cat mai bine ceea ce dorim sa simulam.
* Incercarea dezvoltari unei simulari cat mai aproape de realitate cand un anumit client isi alege aleator casa la care se pune si sta in magazin un timp mai indelungat.
* Alegerea unui numar care determina numarul maxim de clienti la o coada i-ar daca acesta este depasit sa se deschida o noua casa la care sa se puna alti clienti.

**7. Bibliografie**

* <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* <http://www.tutorialspoint.com/java/util/timer_schedule_period.htm>
* <https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>
* <http://javahash.com/java-concurrency-future-callable-executor-example/>
* <https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>
* <https://java.net/downloads/jdf/Cristian_Frasinaru-Curs_practic_de_Java.pdf>
* <http://www.tutorialspoint.com/java/java_multithreading.htm>