Implementarea unei aplicatii de simulare a evolutiei cozilor ce urmareste timpul de asteptare

**Cuprins**

Obiectivul temei…………………………………………………………………………………………………3

Analiza problemei……………………………………………………………………………………………….3

Diagrama UML a claselor…………………………………………………………………………………….4

Implementare……………………………………………………………………………………………………5-9

Rezultate……………………………………………………………………………………………………………..9

Concluzii……………………………………………………………………………………………………………11

Dezvoltari ulterioare………………………………………………………………………………………….11

Bibliografie…………………………………………………………………………………………………………11

1. Obiectivul temei:

Proiectul are ca scop simularea unei situatii reale de utilizare a cozilor si anume folosirea acestor cozi intr-un magazin, adica la casele de marcat. In acest fel se vor simula situatii reale de sosire a clientilor la casele de marcat deschise si de procesarea comenzilor acestora.

Proiectul l-am realizat astfel: se creeaza un numar total de cozi care este primit ca si parametru de intrare din interfata grafica si am implementat metode specifice pentru generarea random a unor clienti in functie de timpul de sosire + timpul de service si i-am asezat pe acestia la cozi in functie de timpul de asteptare.

Obiectivul principal al unei cozi este de a oferi un loc pentru client, un loc unde trebuie sa astepte pana in momentul in care ii vine randul la casa de marcat. Timpul acesta de asteptare difera de la client la client in functie de cati clienti sunt in fata lui si de timpul de service al acestora. Aplicatia creata simuleaza generarea unui numar de clienti care sosesc la casele de marcat pentru un serviciu, introducerea acestora in cozi, fiecare coada reprezentand un thread, timpul de sosire si timpul de asteptare al acestora. In momentul in care un client se aseaza la o coada, timpul de servire al acesteia creste, in momentul in care acesta paraseste coada, timpul de asteptare al cozii scade astfel putandu-se face ordonarea cozilor in functie de asteptare. Prin acest procedeu ne asiguram ca fiecare client ajunge la o coada unde trebuie sa astepte cel mai putin dintre toate cozile disponibile.

Cozile pot fi observate atat in lumea reala cat si in lumea modelelor . Obiectivul principal al unei cozi este de a furniza un loc de asteptare pentru un client inainte de a fi servit . Managementul unui sistem bazat pe cozi are ca si obiectiv reducerea timpului de asteptare al clientilor care stau la coada pentru a fi serviti . Unul din modurile de a reduce timpul de asteptare este de a adauga mai multe case de marca, mai bine zis de a adauga mai multe cozi ( se considera ca fiecare coada are cate un procesor asociat) , dar aceasta metoda creste costul pe care treubuie sa-l suporte furnizorul.

1. Analiza problemei

Am implementat proiectul folosind 5 clase. Aceste clase se pot imparti in 2 categorii: clase care ne ajuta sa simulam evolutia cozilor (Client, Coada, Magazin) si clase care ne ajuta la implementarea interfetei grafice (GUI, App – care de fapt este main-ul).

Clasa GUI este clasa care implementeaza interfata grafica si cu ajutorul ei putem prelucra datele. Datele de intrare se introduc cu ajutorul interfetei grafice, iar datele de iesire vor fi afisate tot in interfata grafica, cu ajutorul unor text-field-uri. Datele de intrare sunt reprezentate sub forma unor valori de tip integer, iar acestea sunt: timpul minim de sosire, timpul maxim de sosire, timpul minim de servire, timpul maxim de servire, numarul total de clienti si numarul total de case de marcat (thread-uri). In interfata grafica se gaseste si un buton START cu ajutorul caruia pornim executia programului.

1. Diagrama UML a claselor

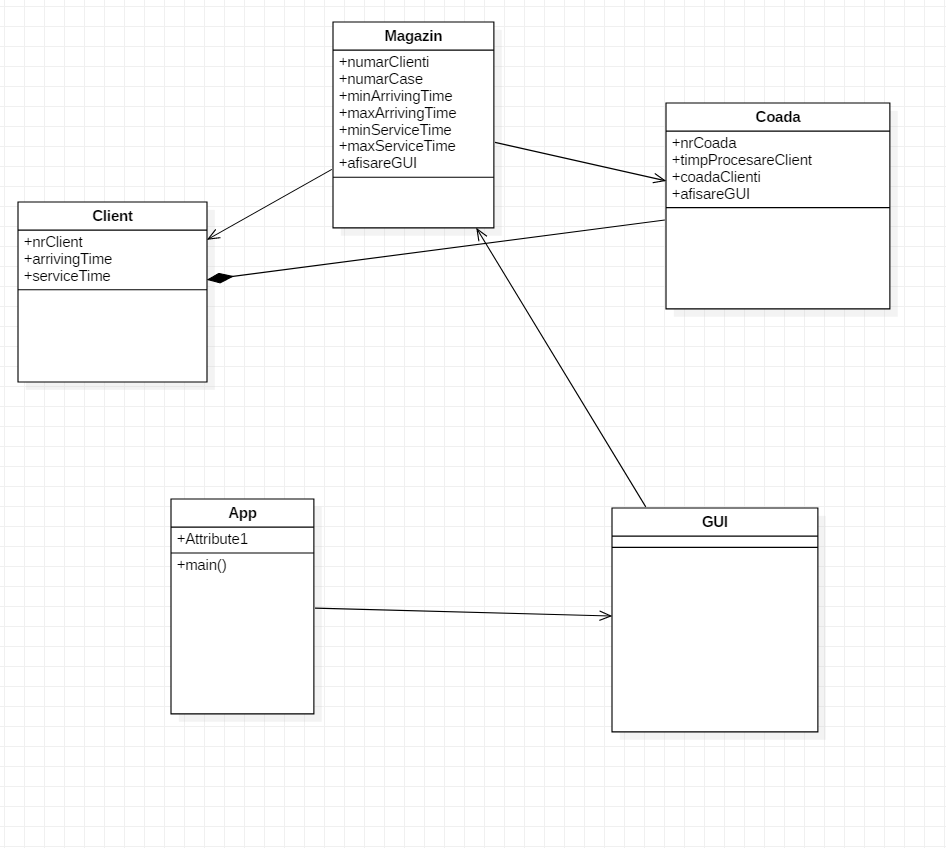
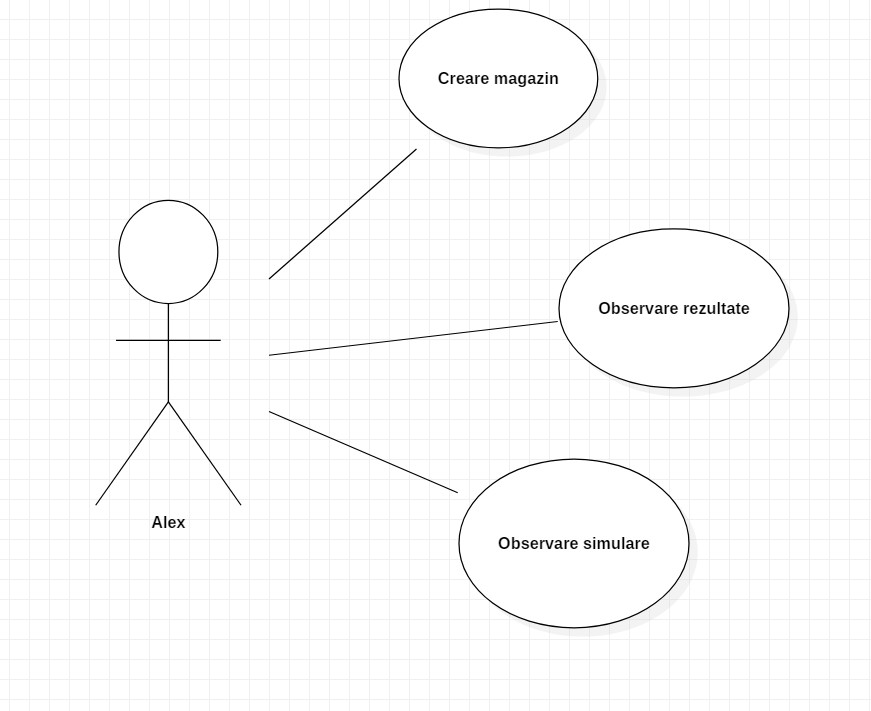


Diagrama Use-Case



1. Implementare

**Clasa Client**

private int nrClient;

private int arrivingTime;

private int serviceTime;

Aceasta clasa are 3 campuri: nrClient, arrivingTime, serviceTime care reprezinta numarul clientului, timpul de sosire si timpul de servire.

public Client(int n1, int a1, int s1) {

this.nrClient = n1;

this.arrivingTime = a1;

this.serviceTime = s1;

} Constructorul clasei Client initializeaza cele 3 campuri pe masura ce se instantiaza un client. In aceasta clasa se mai afla metodele numite get-ere si set-ere cu ajutorul carora setam sau aflam unul dintre campurile clientului.

**Clasa Coada**

Aceasta clasa implementeaza coada propriu-zisa. Ea contine un vector de clienti din care se scot sau se adauga client pe masura ce programul ruleaza.

public Coada(int nr, JTextArea alex)

Constructorul acestei clase initializeaza numarul cozii la variabila nr primita ca si parametru, creeaza un nou vector de clienti, seteaza timpul de procesoare al cozii la 0 si initializeaza variabila de tipul JTextArea alex care va fi folosita pentru afisarea rezultatelor in interfata grafica.

Clasa Coada dispune de mai multe metode, primele dintre acestea fiind metodele numite set-ere si get-ere.

public synchronized void addClient(Client c) throws InterruptedException

Metoda numita sugestiv addClient este metoda cu ajutorul careia se introduce un nou client in vectorul de clienti, adica in coada. In momentul in care un nou client este adaugat, timpul de procesare al cozii se aduna cu timpul de service al clientului, iar dupa acesta se cheama metoda notifyAll() care anunta restul thread-urilor ca si-a facut treaba.

public boolean verifyCoada()

Aceasta metoda numita verifyCoada() verifica daca vectorul de client numit coada este gol sau nu. In cazul in care coada este goala ne returneaza true, in rest false.

public int timpService()

Aceasta metoda este folosita pentru a ne returna timpul de service al primului client din coada, timp care va fi folosit in metoda run pentru a sti cat trebuie sa astepte thread-ul pana se va realiza servirea primului client.

public synchronized void removeClient() throws InterruptedException

Dupa cum spune si numele, aceasta metoda este folosita pentru a scoate un client din coada. Pentru inceput, in metoda se verifica daca coada este goala. In cazul in care coada este goala, nu are client in ea, se asteapta. Daca coada are clienti in ea, cu ajutorul unui String numit s afisam prin intermediul interfetei grafice numele cozii si toti clientii care sunt la momentul respective in aceasta. Aflarea clientilor existenti in coada este foarte simpla, parcurgandu-se intreaga coada cu ajutorul unui for. Dupa aceasta, se face delete la primul client din coada iar timpul de asteptare al cozii scade si el.

public void run()

Aceasta metoda este folosita pentru rularea thread-ului, adica a cozii. Inainte sa intram in bucla infinita realizata cu ajutorul lui while(true), ii dam cozii un sleep de 100 de ms ca sa astepte sa fie populate primele cozi si abia apoi sa inceapa sa scoata clientii din acestea in functie de timpul de service. Cu ajutorul variabilei timp, care de altfel este timpul de service al primului client din coada, facem ca thread-ul nostru sa-si inceteze executia o perioada de timp. Acest lucru se face inmultind variabil timp cu 1000 ms, adica cu o secunda, in acest fel asteptand sa treaca timpul de service al primului client din coada. In cazul in care metoda run genereaza o eroare m-ama asigurat ca aceasta este prinsa prin intermediul unui try-catch si afisata in consola.

**Clasa Magazin**

Aceasta clasa este foarte importanta si o consider creierul proiectului. Ea extinde clasa Thread si are destul de multe campuri, dar necesare, cum ar fi : numarul de clienti, numarul de case de marcat, numarul de cozi, timpul minim de sosire, timpul maxim de sosire, timpul minim de service, timpul maxim de service si un camp JTextArea cu ajutorul caruia vom afisa rezultatele in interfata grafica.

public Magazin(int n1, int n2, Coada coziMagazin[], int m1, int m2, int m3, int m4, JTextArea afisareGUI)

Cu ajutorul acestui constructor initializam toate valorile de care avem nevoie si cream un numar de case de marcat egal cu numarul primit ca si parametru.

public int selectareCoadaClient() throws InterruptedException

Cu ajutorul acestei metode reusim sa selectam coada in care va trebui sa fie introdus noul client sosit. Aceasta selectie se face in functie de timpul de procesare al cozii, adica selectam coada cu cel mai mic timp de procesare dintre toate cozile. In acest fel ne asiguram ca fiecare client se aseaza la coada cu cel mai mic timp de asteptare. Aceasta metoda nu este una foarte eficienta deoarece de fiecare data cand este apleata parcurge cu ajutorul unui for toate cozile existente cautand coada cu valoarea minima a campului timpDeAsteptare. S-ar fi putut face intr-un mod mult mai efficient cum ar fi, de exemplu, un vector in care sa avem toti timpii cozilor.

public void run() {

try {

String s = "";

double avgArr=0;

double avgSrv = 0;

int i = 1;

while (i <= numarClienti) {

Random ran = new Random();

Client c = new Client(i, minArrivingTime + ran.nextInt(maxArrivingTime - minArrivingTime),

minServiceTime + ran.nextInt(maxServiceTime - minServiceTime));

int selectareCasaAleasa = selectareCoadaClient();

//System.out.println(selectareCasaAleasa);

//System.out.println("Clientul " + i + " add la casa " + Integer.toString(selectareCasaAleasa + 1));

coziMagazin[selectareCasaAleasa].addClient(c);

avgArr += c.getArrivingTime();

avgSrv += c.getServiceTime();

s = "\nClientul " + i + " add la casa " + Integer.*toString*(selectareCasaAleasa + 1);

afisareGUI.append(s);

*sleep*((long) Math.*random*());

i++;

}

s="\nAverrage arriving = ";

System.***out***.println("Average arriving = "+ avgArr / (numarClienti-1));

s += avgArr / (numarClienti - 1) ;

afisareGUI.append(s);

s="\nAverrage servicing = ";

System.***out***.println("Average servicing = "+ avgSrv / (numarClienti-1));

s += avgSrv / (numarClienti - 1);

afisareGUI.append(s);

} catch (InterruptedException e) {

System.***out***.println(e.toString());

}

}

Cu ajutorul unui while ne cream toti clientii, dar nu oricum, ci cu ajutorul functiei Random. Asta inseamna ca toti clientii vor fi diferiti, avand nu doar ID-urile diferite ci si timpii de sosire respective de astepare. Dupa crearea clientului, se apeleaza metoda selectareCoadaClient() cu ajutorul careia repartizam noul client sosit la coada cu cel mai mic timp de astepare. Dupa aceasta, adunam in variabilele locale avgArr si avgSrv timpii de sosire si de service ai clientului pentru ca mai apoi sa calculam timpii medii de asteptare ai tuturor clientilor. Mai apoi thread-ul primeste un sleep random dupa care genereaza creearea unui nou client random si tot asa pana ne-am generat toti clientii.

**Clasa GUI**

Aceasta clasa este foarte stufoasa ocupand 168 de linii de cod. Scopul acesteia este de a crea interfata grafica si de a face legatura cu programul nostrum.

Object source = arg0.getSource();

try {

if(source == start) {

String s1 = minArriving.getText();

int n1 = Integer.*parseInt*(s1.toString());

String s2 = maxArriving.getText();

int n2 = Integer.*parseInt*(s2.toString());

String s3 = minServicing.getText();

int n3 = Integer.*parseInt*(s3.toString());

String s4 = maxServicing.getText();

int n4 = Integer.*parseInt*(s4.toString());

String s5 = noClients.getText();

int n5 = Integer.*parseInt*(s5.toString());

String s6 = noCase.getText();

int n6 = Integer.*parseInt*(s6.toString());

int i;

int numarCase=n6;

Coada c[]=new Coada[numarCase];

for(i=0;i<numarCase;i++)

{

c[i]= new Coada(i,rezultat);

c[i].start();

}

m = new Magazin(n5,n6,c,n1,n2,n3,n4,rezultat);

m.start();

}

}catch(Exception e) {

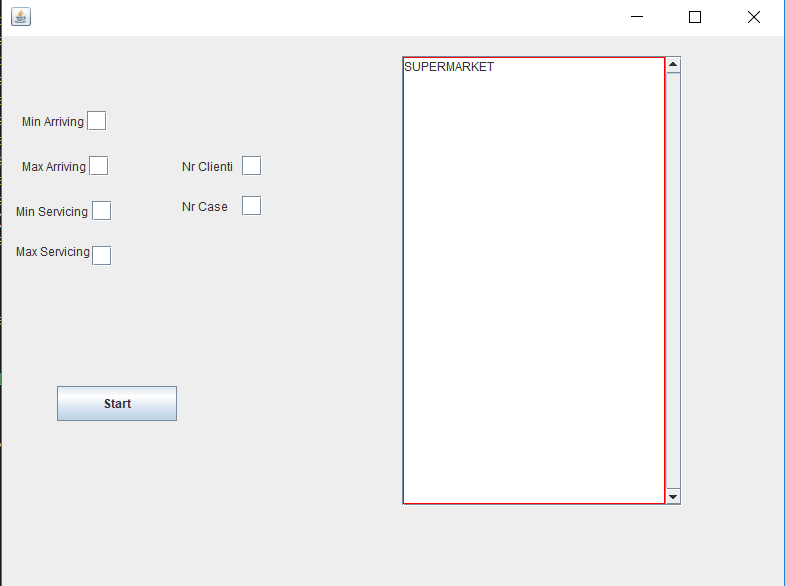
System.***out***.println("Eroare la introducerea datelor ! ");

}

}

Aceasta metoda preia valorile introduse de noi in campurile disponibile din interfata grafica, le parseaza si mai apoi porneste executia propriu-zisa a programului urmand ca rezultatele sa se afiseze in interfata grafica si nu in consola. Mai precis se declara o variabila numarCase care va prima valoarea introdusa de noi ca fiind numarul total de case de marcat din magazine si pe baza acestei variabile se vor instantia numarCase cozi urmand ca mai apoi sa se introduca respectiv scoata clienti din ele.

1. Rezultate



Asa va arata interfata grafica in momentul in care se va executa programul nostrum. Se pot observa campurile precum timpul minim de asteptare, timpul maxim de asteptare, timpul minim de service al fiecarui client, timpul maxim de service al fiecarui client respective numarul total de client si case. In momentul in care cele 6 campuri vor fi completate, se va apasa butonul de Start si va incepe executia programului. In momentul in care in text field-uri se vor introduce valori gresite precum numere reale, caractere sau orice altceva in afara de numere intregi si se va apasa butonul de Start, in consola se va afisa o eroare si executia programului nu va incepe.

1. Concluzii

In concluzie, simularea evolutiei unei cozi pare o problema destul de usoara la inceput, dar dificultatile nu inceteaza sa apara pe parcurs, mai ales ca trebuie folosite conceptele programarii orientate pe obiect. Cert este ca a fost o tema foarte grea, nu imposibila, dar grea.

Limbajul Java ne ofera o perspectiva foarte larga privind simularea evolutiei cozilor cu privire la constructia de clase si metodele dezvoltate.

1. Dezvoltari ulterioare

Programul implementat nu este foarte efficient. Pe viitor, ar trebui sa se lucreze la eficienta programului si la unele metode cum ar fi metoda de selectare a cozii cu timpul minim de asteptare, generarea random a clientilor si infrumusetarea interfetei grafice.

Aplicatia creata ar putea fi imbunatatita creeandu-se o alta metoda de oprire si remodelarea interfetei grafice.

1. Bibliografie

* http://stackoverflow.com/questions/
* http://docs.oracle.com/en/
* http://www.tutorialspoint.com/java/