DOCUMENTATIE TEMA 2

MAGAZIN

Nume si prenume: Fagadar Ionela-Catalina

Grupa: 30226

Profesor Laborator: Marin Oana

Contents

[**1.Obiectivul temei** 3](#_Toc5189433)

[**2.Analiza problemei, modelare, scenarii, cazuri de utilizare** 3](#_Toc5189434)

[**3.Proiectare** 3](#_Toc5189435)

[**3.1. Decizii de proiectare** 4](#_Toc5189436)

[**3.2. Diagrame UML** 4](#_Toc5189437)

[**3.3. Structuri de date** 5](#_Toc5189438)

[**3.4. Proiectare clase** 5](#_Toc5189439)

[**3.5. Proiectare interfete** 6](#_Toc5189440)

[**3.6. Relatii** 6](#_Toc5189441)

[**3.7. Pachete** 6](#_Toc5189442)

[**3.8. Algoritmi** 6](#_Toc5189443)

[**3.8.1. Algoritmul de adaugare in coada** 6](#_Toc5189444)

[**3.8.2. Algoritmul de generare al clientilor** 6](#_Toc5189445)

[**3.8.3. Algoritmul pentru metoda run() a Simularii** 7](#_Toc5189446)

[**4. Implementare** 7](#_Toc5189447)

[**4.1. Clasa Casa -Pachet CasaSiClient** 7](#_Toc5189448)

[**4.2. Clasa Client - Pachet CasaSiClient** 7](#_Toc5189449)

[**4.3. Clasa Magazincontroller – Pachet Controller** 7](#_Toc5189450)

[**4.4. Clasa MagazinModel – Pachet Model** 7](#_Toc5189451)

[**4.5. Clasa MagazinMVC -Pachet MVC** 8](#_Toc5189452)

[**4.6. Clasa MagazinView -Pachet View** 8](#_Toc5189453)

[**4.7. Clasa Simulare -Pachet Simulare** 8](#_Toc5189454)

[**5.Rezultate** 9](#_Toc5189455)

[**6.Concluzii si dezvoltari ulterioare** 9](#_Toc5189456)

[**7.Bibliografie** 9](#_Toc5189457)

# **1****.Obiectivul temei**

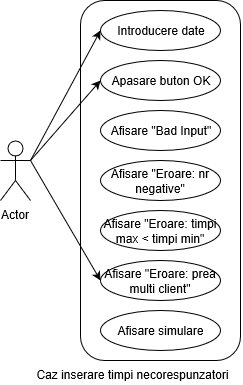
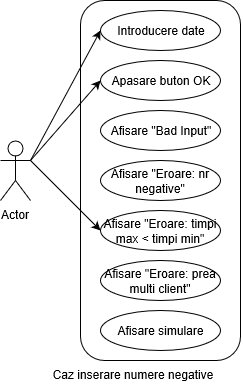
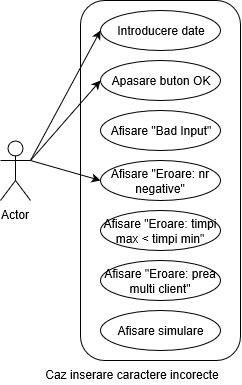
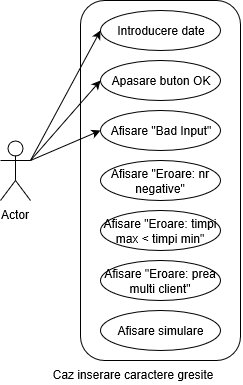
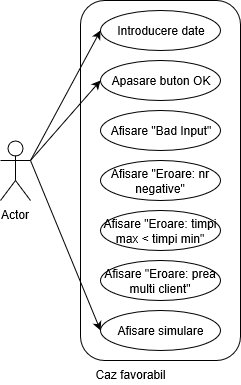
Propuneti, proiectati si implementati un aplicatie de simulare facuta sa analizeze cea mai buna pozitie de punere la coada pentru a determina si minimiza timpul de asteptare al clientilor.

|  |  |  |
| --- | --- | --- |
| Obiectiv | Descriere | Capitol |
| dezvoltarea de use case-uri si scenarii |  | 2 |
| alegerea structurilor de date |  | 3 |
| impartirea pe clase |  | 3 |
| dezvoltarea algoritmilor |  | 3 |
| implementarea solutiei |  | 4 |

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

Programul este conceput astfel incat, utilizatorul introduce un timp minim sosire client si timp maxim sosire client, timp in care clientii pot ajunge in magazin. Inainte de aceasta perioada, sau dupa, clientii nu mai sosesc in magazin. Se mai introduc timpii necesari minimi si maximi, timpi ce semnifica cat pot dura cumparaturile fiecarui client. O alta variabila introdusa de utilizator este numarul de case, ce semnifica si nr de cozi la care se pot aranja clientii. Numarul acestora este tot o variabila introdusa de utilizator. Intervalul de Simulare se introduce de catre utilizator,semnificand un numar de secunde pe parcursul carora simularea se va efectua, iar acesta nu poate sa fie mai mic decat timpul maxim de sosire a clientilor. Se afiseaza exceptii in cazul in care: Se introduc caractere ce nu sunt numere, de ex „1a”, se introduc timpi negativi, sau timp de simulare nul, numar case nul sau numar clienti nul, afisandu-se mesajul „Eroare: nr negative”, se introduce timpul de sosire maxim mai mic decat timpul de sosire minim, sau timpul de procesare maxim mai mic decat timpul de procesare minim.In acest caz, apare mesajul „Eroare: timpi max < timpi min”. O alta exceptie apare in momentul in care introducem un numar de Clienti mai mare decat intervalul de sosire dat, deoarece s-a ales ca doi clienti sa nu poata ajunge in acelasi timp in magazin, totul intamplandu-se sincron: „Eroare: prea multi clienti”.

Diagrama use-case a acestui proiect este:



# 

# **3.Proiectare**

## **3.1. Decizii de proiectare**

Am ales sa proiectez acest proiect in modul MVC: Model View Controller, impartindu-l in 6 pachete: MVC, Model, Controller, View, CasaSiClient, Simulare. Initial, am ales ca simularea sa nu inceapa pana nu se introduc toti timpii corect, si se apasa butonul „OK”(View+Controller). Acesta face ca simularea sa porneasca, un numar de secunde egal cu intervalul de simulare introdus. Logica aplicatiei se regaseste in pachetul Model, in clasa MagazinModel, si in pachetul Simulare in clasa Simulare. Am decis ca, aceasta aplicatie sa foloseasca mai multe fire de executie, dat fiind faptul ca trebuia ca procesele de introducere a clientilor in cozi, si de scoatere a acestora, sa fie sincrone cu timpul simularii, si pentru toate casele sa se execute in acelasi timp . Un fir de executie este folosit in general acolo unde se vrea sa se intample mai multe lucruri in acelasi timp (in cazul nostru, casele). Astfel, am ales ca clasa Simulare sa fie un fir de executie, care face ca totul sa se execute din secunda in secunda, iar celelalte fire de executie sa fie cele nrCase create.

## **3.2. Diagrame UML**

Diagrama de clase a acestui proiect:

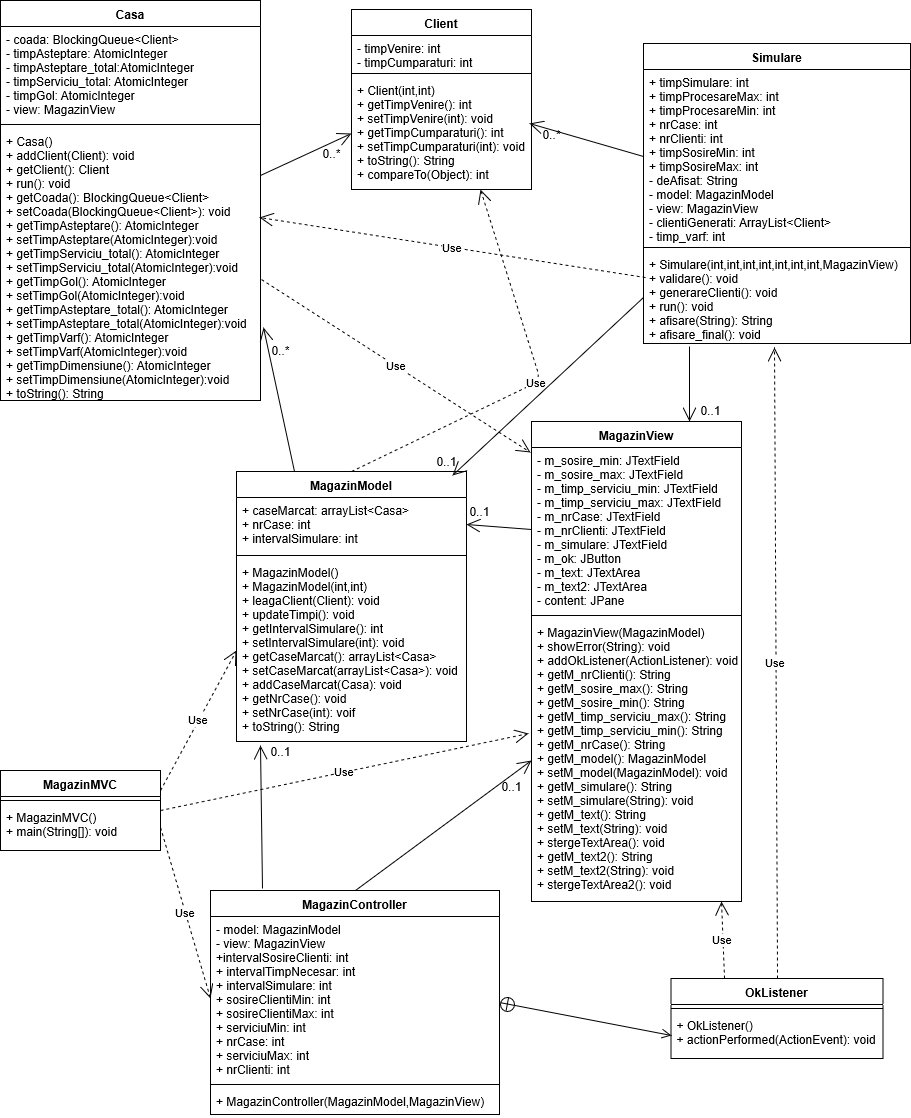
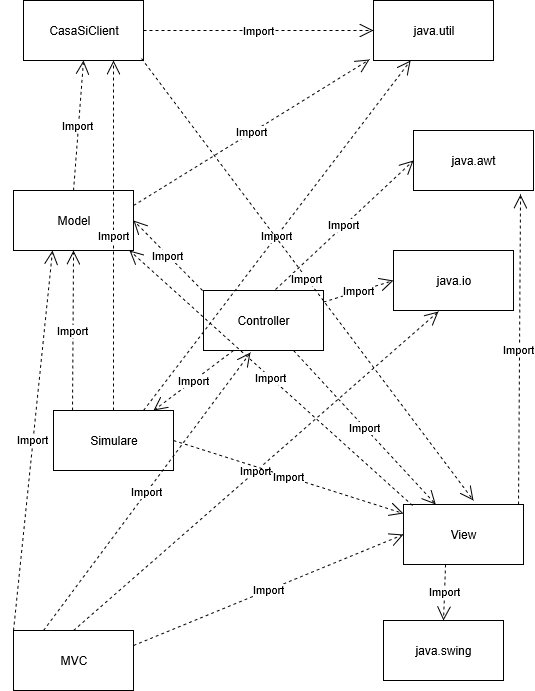


Diagrama de pachete a acestui proiect:



## **3.3. Structuri de date**

Structura de date folosita in principal in acest proiect este BlockingQueue, folosita in cadrul clasei Casa. Aceasta contine o serie de clienti care isi asteapta procesarea in coada. S-a ales aceasta structura de date datorita faptului ca este o structura sincrona. O alta structura de date folosita, este Lista ( ArrayList), folosita in generarea Clientilor in logica Simularii.

## **3.4. Proiectare clase**

Se foloseste o clasa pentru afisarea in timp real a cozilor, pentru primirea de timpi si numere folositoare simularii, notificarea sosirii si plecarii de la coada a clientilor si pentru rezultatele simularii in cazul mediu ( Clasa View). O alta clasa se foloseste pentru a controla datele introduse si pentru a initializa si porni simularea. (Clasa Controller). Clasa Model este responsabila pentru logica aplicatiei, in mare parte, in aceasta facandu-se legatura dintre coada si clienti, cat si pentru actualizarea timpilor de caz mediu. Clasa Casa reprezinta conceptul de casa de marcat din lumea reala, aceasta avand o coada la care asteapta clientii, si cativa timpi care se vor folosi la cazul mediu. Clasa Client cuprinde timpul de sosire si cel de procesare al fiecarui client, si reprezinta conceptul de Client din lumea reala. Cea mai importanta este clasa Simulare, care genereaza clientii ce sosesc la magazin, si care gestioneaza introducerea acestora la cozi, astfel incat timpul de asteptare al fiecaruia sa fie minim.

## **3.5. Proiectare interfete**

Singura interfata folosita este interfata Runnable, interfata caracteristica Thread-urilor (in Clasa Simulare si Casa).

## **3.6. Relatii**

Relatiile folosite in acest proiect sunt relatii de mostenire intre clasa MagazinView si clasa JFrame (clasa MagazinView extinde clasa JFrame), relatii de implementare intre clasa Casa si clasa Simulare cu interfata Runnable( Clasa Casa implementeaza Runnable; clasa Simulare implementeaza Runnable), si clasa Client cu interfata Comparable ( Clasa Client implementeaza Comparable).

## **3.7. Pachete**

Proiectul e structurat in 6 pachete:MVC, Model, Controller, View, CasaSiClient,Simulare. Pachetul Controller contine clasa ce determina controlul interfetei cu utilizatorul (contine clasele MagazinController).Pachetul Model ( contine clasa MagazinModel) si pachetul Simulare( contine Clasa Simulare) determina logica aplicatiei. Pachetul View contine o singura clasa ce determina interfata grafica a programului, mai exact modul cum sunt afisate butoanele, casetele text, etichetele ( contine clasa MagazinView). Pachetul CasaSiClient contine elementele necesare aplicatiei ( contine clasele Client si Casa), iar pachetul MVC contine doar o clasa MagazinMVC ce contine metoda main de unde se incepe executia programului.

S-au importat pachete implicite java, ca java.util.concurrent.\*; java.awt.event.ActionEvent; java.awt.event.ActionListener; java.io.IOException;java.util.\*;java.util.concurrent.atomic.AtomicInteger;java.util.Collections;javax.swing.\*; java.awt.\*; java.awt.event.ActionListener;

## **3.8. Algoritmi**

Algoritmii utilizati pentru acest proiect sunt algoritmii utilizati pentru operatii:adaugare in coada si cel de generare al clientilor. Se mai poate adauga si metoda de run() a simularii, care da in mare parte logica aplicatiei.

## **3.8.1. Algoritmul de adaugare in coada**

In acest algoritm, se adauga la coada cea mai preferabila, adica, la coada la care clientul ar astepta cel mai putin. Astfel, se initializeaza o variablia min cu cel mai mare numar intreg reprezentabil (MAX\_VALUE). Clasa casa este construita in asa fel incat, fiecare Casa are un timp de asteptare, adica un timp care semnifica durata de timp necesara pentru client inaintea procesarii. Aceasta este calculata insumand timpii de procesare ramasi a fiecarui client de la coada. Se cauda coada cu timpul minim de procesare, iar apoi, clientul se introduce la coada.

## **3.8.2. Algoritmul de generare al clientilor**

In acest algoritm se si valideaza unele date primite de la utilizator. Asfel, daca numarul clientilor este mai mare decat intervalul de sosire, se arunca o exceptie. Se remarca un caz exceptional, caz in care numarul de clienti este egal cu intervalul de sosire. Astfel, clientii ar trebui sa apara din secunda in secunda, unul cate unul. Aici, pentru fiecare client se genereaza aleatoriu un timp de cumparaturi, iar timpul de venire se incrementeaza de la client la client. Clientul se adauga intr-o lista de Clienti Generati. In cazul in care nu ne aflam in cazurile precedente, se genereaza un numar de clienti egal cu nrClienti, astfel: se calculeaza aleatoriu un timp de venire si un timp de cumparaturi. Se cauta in lista de clienti generati daca exista un client generat cu acelasi timp de venire, iar in caz afirmativ se reia calculul aleatoriu. In caz negativ, clientul se adauga in lista si se incrementeaza numarul de clienti. Se sorteaza lista de clienti generati, dupa timpul de sosire, pentru a-l avea pe clientul ce ajunge primul in capul listei.

## **3.8.3. Algoritmul pentru metoda run() a Simularii**

Acest algoritm face ca totul sa se execute pas cu pas in acest program. Astfel, se incepe de la timpul 1, iar pentru un timp egal cu timpul de procesare, se procedeaza in fiecare secunda dupa cum urmeaza: Se verifica daca mai sunt clienti generati ce nu au fost pusi la coada. Se ia primul client, daca are timpul de venire egal cu timpul curent. Se apeleaza metoda de adaugare in coada ce are la baza algoritmul descris la sectiunea 3.8.1, iar clientul generat se sterge din capul listei. Timpii de caz mediu se actualizeaza. Se afiseaza structura cozii. Se incrementeaza timpul curent, iar thread-ul corespunzator simularii se pune pe sleep pentru o secunda. La finalul simularii, se afiseaza timpii cazului mediu.

## **3.8.4. Algoritmul pentru metoda run() a Casei**

Se procedeaza in felul urmator pentru toate Thread-urile corespunzatoare clasei Casa deschise: Se verifica daca coada acestora este goala. In cazul in care nu este goala, se extrage primul client din coada. Se decrementeaza timpul de asteptare al acesteia, cat si timpul de cumparaturi al clientului. Daca timpul de cumparaturi ajunge la zero, clientul se sterge din coada, se afiseaza textul „Un client a iesit din coada”. La sfarsit, Thread-ul se pune pe sleep o secunda, pentru a-l sincroniza cu restul simularii.

# **4. Implementare**

## **4.1. Clasa Casa -Pachet CasaSiClient**

Atribute: coada (BlockingQueue pentru sincronizare), timpAsteptare (pentru afisarea in timp real a timpului pana cand se elibereaza coada), timpAsteptare\_total , timpServiciu\_total, timpGol.

Constructori , Getter, Setter.

Metoda addClient(Client c) care adauga clientul si actualizeaza timpul de asteptare, insumandu-l cu timpul de cumparaturi al clientului. Metoda run() foloseste algoritmul prezentat in sectiunea 3.8.4 .Metoda toString() pentru afisare.

## **4.2. Clasa Client - Pachet CasaSiClient**

Atribute: timpVenire, timpCumparaturi

Constructori , Getter, Setter, metoda toString() pentru afisare, si metoda CompareTo(Object arg0) folosita pentru sortare.

## **4.3. Clasa Magazincontroller – Pachet Controller**

Atribute: model, view, intervalSosireClienti, intervalTimpNecesar, intervalSimulare, sosireClientiMin, sosireClientiMax, serviciuMin, nrCase, serviciuMax, nrClienti.

Constructori.

Aceasta clasa mai contine o alta clasa interna, OKListener care, la apasarea butonului OK in interfata grafica, extrage datele de intrare furnizate de utilizator prin interfata grafica, aruncand exceptii in cazul in care formatul introdus nu este unul acceptat, apoi porneste simularea.

## **4.4. Clasa MagazinModel – Pachet Model**

Atribute: caseMarcat, nrCase, intervalSimulare.

Constructori, Getter, Setter, metoda toString() pentru afisare.

Metode sincronizate: leagaClient (Client c) care calculeaza locul clientului la coada cea mai favorabila din punct de vedere al timpului de asteptare si apeleaza metoda addClient din Clasa Casa, updateTimpi() care actualizeaza timpii in care casa era goala, cat si timpii in care clientii asteapta la coasa (timpi de asteptare).

## **4.5. Clasa MagazinMVC -Pachet MVC**

Metoda main() care leaga Model de View si de Controller, iar de aici incepe aplicatia.

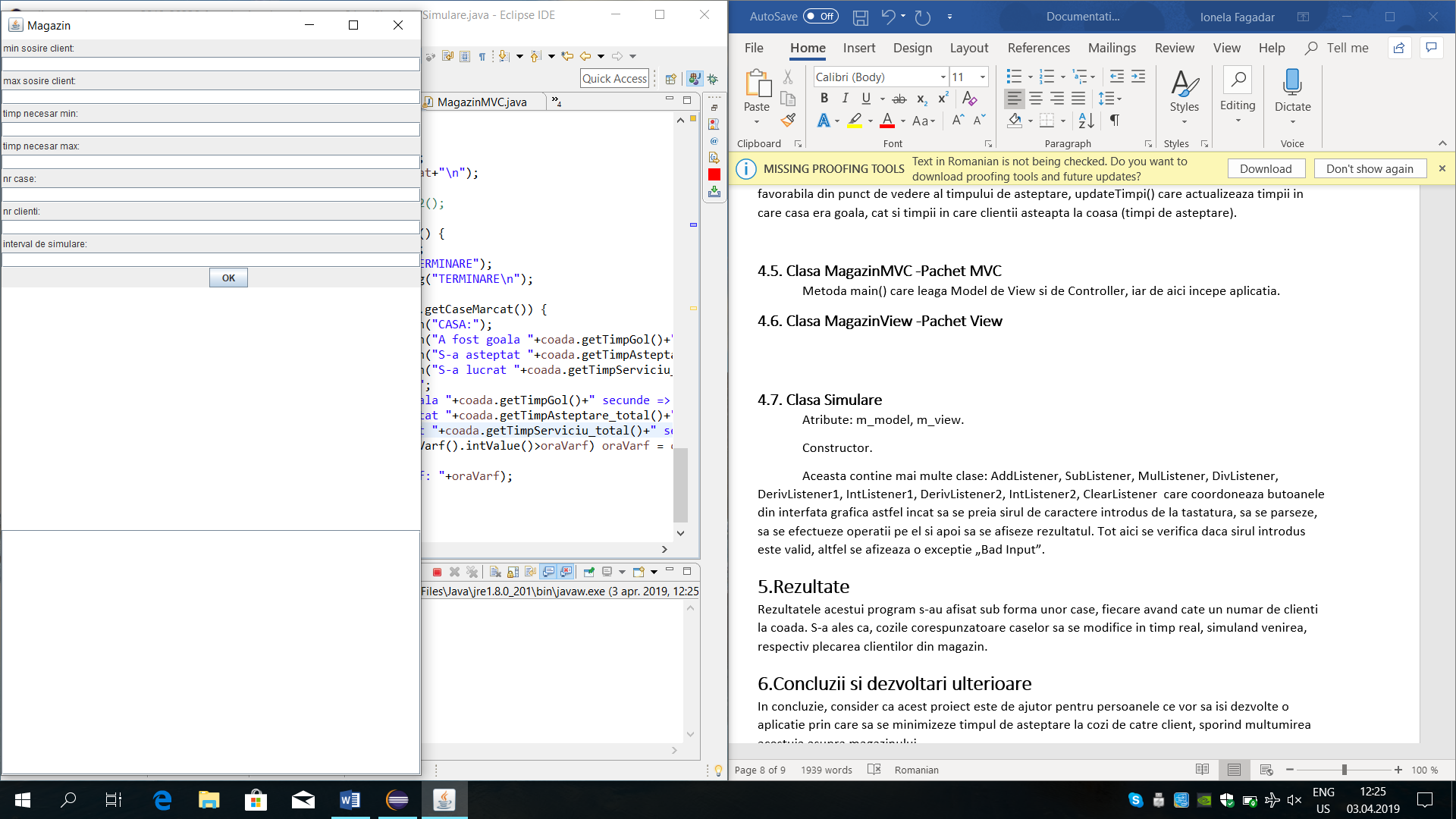
## **4.6. Clasa MagazinView -Pachet View**

Atribute: m\_sosire\_min, m\_sosire\_max, m\_timp\_serviciu\_min, m\_timp\_serviciu\_max, m\_nrCase, m\_nrClienti, m\_simulare, m\_ok, m\_model, m\_text, m\_text2, content – atribute utilizate pentru etichete, utilizate pentru casete text in care se introduc datele de intrare, butonul care initializeaza simularea, casete text care afiseaza simularea in care clientii se adauga la cozi sau se scot.

Constructorul e utlizat pentru a adauga casetele text, butoanele, etichetele in clasa View, afisandu-le pe ecran.

Getter, Setter, metoda ShowError(String errMessage) pentru a afisa un mesaj in momentul introducerii de catre utilizator a unui format inacceptat in aplicatie, metoda addListener(ActionListener ok) pentru a-i atribui butonului o functie atunci cand este apasat. Metodele stergeTextArea(),stergeTextArea2() care sterg continutul lui textArea.

Caseta corespunzatoare view-ului arata in urmatorul fel:



Aici se afiseaza cozile la case in timp real

Aici se afiseaza cand clientii ies sau intra la coada

Aici se introduc datele de intrare necesare

## **4.7. Clasa Simulare -Pachet Simulare**

Atribute: timpSimulare, timpProcesareMax, timpProcesareMin, nrCase, nrClienti, timpSosireMin, timpSosireMax, deAfisat, model, view, clientiGenerati (ArrayList<Client>), timp\_varf.

Constructor care initializeaza atributele, creeaza nrCase de case de marcat (instante de clase si thread-uri), valideaza datele introduse de utilizator prin metoda validare(), genereaza clientii prin metoda generareClienti().

Metoda validare() verifica daca timpii introdusi sunt negativi, sau timpul de simulare este nul, sau numarul de case este nul, sau numarul clientilor este nul. Aceasta mai verifica daca timpul de sosire maxim este mai mic decat cel minim, sau timpul de procesare maxim este mai mic decat cel minim. In aceste cazuri, validare() arunca o exceptie cu un mesaj de eroare, ca „Eroare: nr negative” sau „Eroare: timpi max < timpi min”.

Metoda generareClienti() foloseste algoritmul descris la sectiunea 3.8.2 .

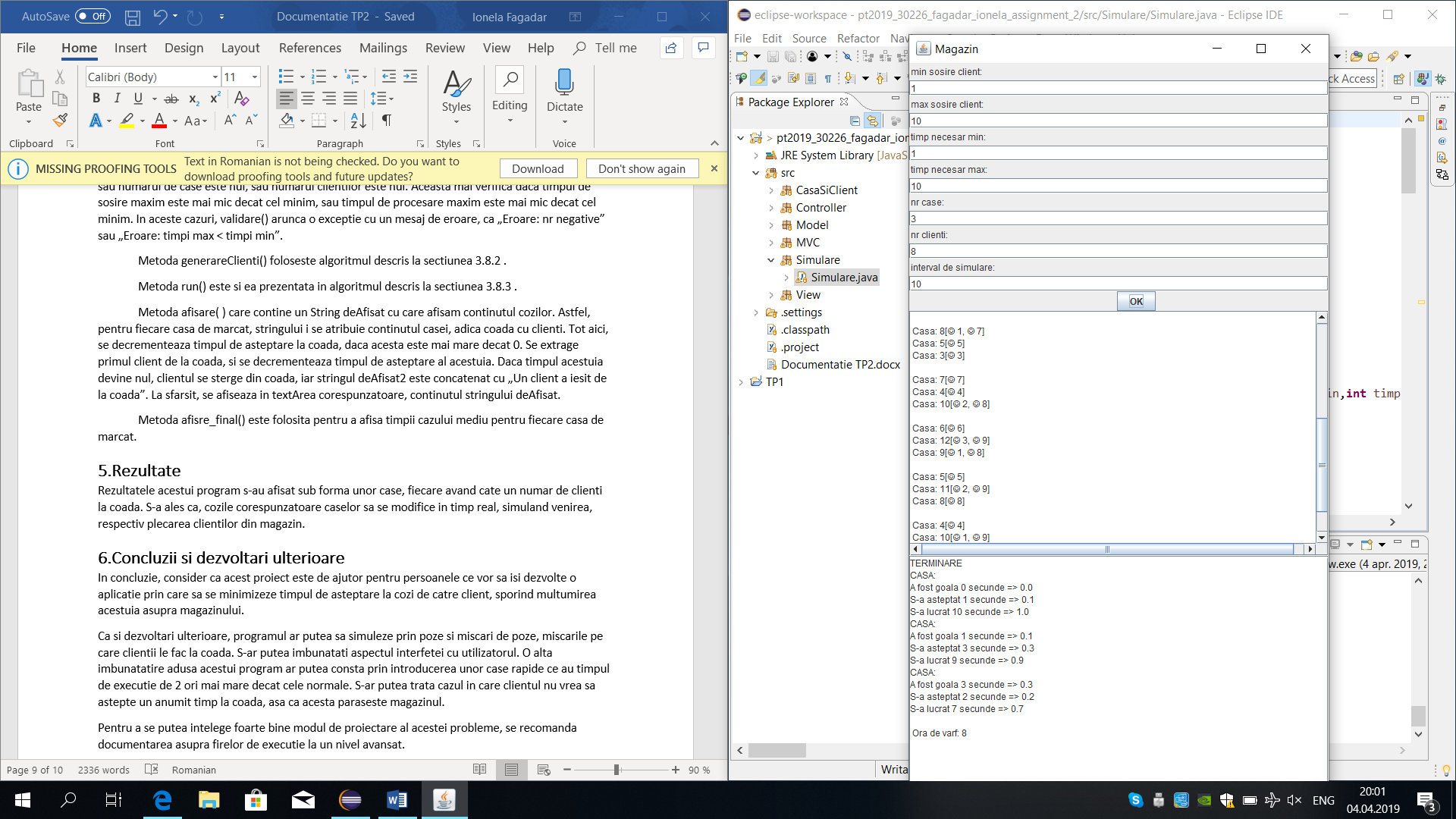
Metoda run() este si ea prezentata in algoritmul descris la sectiunea 3.8.3 .

Metoda afisare( ) care contine un String deAfisat cu care afisam continutul cozilor. Astfel, pentru fiecare casa de marcat, stringului i se atribuie continutul casei, adica coada cu clienti. La sfarsit, se afiseaza in textArea corespunzatoare, continutul stringului deAfisat.

Metoda afisare\_final() este folosita pentru a afisa timpii cazului mediu pentru fiecare casa de marcat.

# **5.Rezultate**

Rezultatele acestui program s-au afisat sub forma unor case, fiecare avand cate un numar de clienti la coada. S-a ales ca, cozile corespunzatoare caselor sa se modifice in timp real, simuland venirea, respectiv plecarea clientilor din magazin.



Timpul de cumparaturi ramas pentru client

Timpul de lucru al casei (suma timpilor de cumparaturi ramasi a tuturor clientilor de la coada)

# **6.Concluzii si dezvoltari ulterioare**

In concluzie, consider ca acest proiect este de ajutor pentru persoanele ce vor sa isi dezvolte o aplicatie prin care sa se minimizeze timpul de asteptare la cozi de catre client, sporind multumirea acestuia asupra magazinului. Totodata, aceasta program poate fi aplicat unui dispozitiv, care ajuta in asezarea clientilor la coada.

Ca si dezvoltari ulterioare, programul ar putea sa simuleze prin poze si miscari de poze, miscarile pe care clientii le fac la coada. S-ar putea imbunatati aspectul interfetei cu utilizatorul. O alta imbunatatire adusa acestui program ar putea consta prin introducerea unor case rapide ce au timpul de executie de 2 ori mai mare decat cele normale. S-ar putea trata cazul in care clientul nu vrea sa astepte un anumit timp la coada, asa ca acesta paraseste magazinul.

Pentru a se putea intelege foarte bine modul de proiectare al acestei probleme, se recomanda documentarea asupra firelor de executie la un nivel avansat.

# **7.Bibliografie**

**Java Concurrency in practice**

* **http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html**
* [**http://www.tutorialspoint.com/java/util/timer\_schedule\_period.htm**](http://www.tutorialspoint.com/java/util/timer_schedule_period.htm)
* **http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-andthreadpoolexecutor.html**
* **http://javahash.com/java-concurrency-future-callable-executor-example/**