***Sistem de simulare al comportamentului in cadrul cozilor***

# Obiective

Creearea unei aplicatii menite sa permita simularea unui sistem bazat pe clienti si respectiv cozi, in vederea analizei timpilor de procesare al comenzilor si minimizarea acestora prin abordarea de diferite strategii si implementarea de algoritmi multiplii cu scopul de a obtine timpul minim de asteptare in functie de scenariul initial propriu zis. Astfel, se identifica urmatoarele obiective secundare in vederea obtinerii unei aplicatii viabile si sustenabile:

|  |  |
| --- | --- |
| Obiective |  |
| Analiza cerintelor functionale  (Capitolul 2) | * Identificarea scenariilor de functionare din punct de vedere al utilizatorului si valorile de intrare introduse de acesta * Identificarea cerintelor functionale: operatiile pe date ce trebuie efectuate de catre sistem * Modelarea procesului de functionare al aplicatiei |
| Proiectarea  (Capitolul 3) | * Structurarea proiectului: impartirea sa in clase si pachete, diagrama UML * Definirea unor structuri de date ce sa faciliteze implementare precum si lizibilitatea codului * Analiza algoritmilor ce trebuie implementati in vederea cresterii performantei |
| Implementarea propriu-zisa (Capitolul 4) | * Punerea in practica a pasilor anterior * Urmarea pasilor de proiectare in mod sistemic pentru evitarea scrierii de cod redundant * Scrierea de cod lisibil si usor de interpretat prin limitarea liniilor de cod (maxim 30 de linii per metoda, respectiv maxim 300 de linii per clasa) |
| Analiza rezultatelor (Capitolul 5) | * Conceperea de teste menite sa verifice fiabilitatea si corectitudinea implementarii tuturor cerintelor funtionale * Interpretarea rezultatelor si descrierea eventualelor erori aparute sau identificate prin acestea |
| Dezvoltare  (Capitolul 6) | * Analiza retrospectiva a proiectului surprinzand dificultatile intalnite * Prezentarea posibilitatilor de dezvoltare ulterioare ale aplicatiei |

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Functionalitatile identificate in urma analizei cerintelor stipulate de catre client sunt:

1. Adaptarea aplicatiei la conditii initiale diferite precum:

* Variatia intervalului de timp dintre aparitia a doi clienti ce trebuie atribuiti unei cozi, prin definirea minimului si maximului intervalului
* Variatia intervalului de timp necesar procesarii comenzii fiecarui client, din nou, prin definirea minimului si maximului intervalului
* Cresterea, respectiv micsorarea numarului de cozi disponibile in vederea gasirii numarului optim, astfel incat timpul de procesare sa se incadreze in limitele propuse si sa nu existe cozi goale
* Alegerea intervalului de timp pentru care se doreste vizualizarea simularii

1. Oferirea unei interfete care sa faciliteze interpretarea simularii:

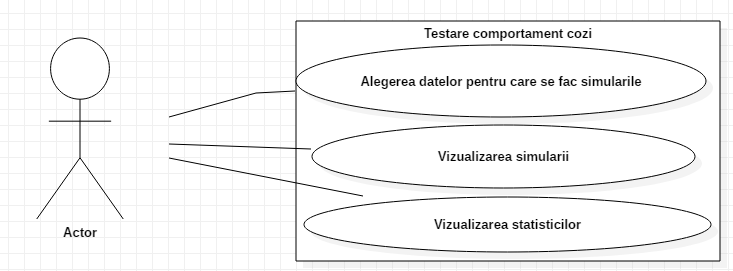
* Simularea cozii prin afisarea clientilor prezenti in ea la un anumite moment de timp
* Afisarea timpului total de procesare ramas al clientilor din fiecare coada
* Scrierea intr-un logger a intrarilor si iesirilor clientilor din cozi, precizandu-se timpul la care au avut loc

1. Calculul de statistici in urma executiei simularii precum:

* Media timpilor de asteptare a fiecarui client la coada
* Timpul maxim inregistrat de asteptare la coada
* „Rush Hour” – momentul de timp in care s-au inregistrat cei mai multi clienti asteptand la coada

1. Oferirea unui grad de generalizare a aplicatiei prin:

* Alegerea intr-un mod aleator a timpului de sosire la coada a clientilor, respectandu-se intervalul ales la deschiderea aplicatiei
* Alegerea intr-un mod aleator a timpului de procesare necesar in satisfacerea fiecarui client, respectandu-se intervalul ales la deschiderea aplicatiei
* Alegerea unei strategii de alocare a clientilor in coada: se alege coada cu cel mai mic timp de asteptare,se alege coada cu cei mai putini clienti etc.



In cele ce urmeaza vor fi prezentate cazurile de utilizare ale aplicatiei:

*Actor principal: Utilizator al aplicatiei (Arhitect, Manager magazin etc.)*

**Use case principal:**

1. Actorul deschide aplicatia
2. Actorul introduce intervalul de timp dintre clienti, intervalul legat de timpul de procesare al clientilor, numarul de cozi si timpul pentru care sa se realizeze simularea (in secunde)
3. Aplicatia verifica input-ul primit
4. Aplicatia lanseaza interfata pentru vizualizarea simularii
5. Simularea incepe, clientii sunt generati si introdusi in coada cu timpul minim de asteptare
6. In log se marcheaza introducerea si scoaterea clientilor din cozi
7. Simularea ajunge la finalul timpului specificat
8. Simularea se opreste
9. Aplicatia lanseaza un nou panou unde se afiseaza statisticile
10. Actorul interpreteaza rezultatele primite din statistici

**Scenarii alternative:**

1. Intervalul de timp dintre clienti este gresit: una dintre valori este negativa sau timpul minim este mai mare in comparatie cel maxim

* Aplicatia transmite actorului ca input-ul este gresit
* Scenariul se intoarce la pasul 2

1. Timpul de procesare al clientilor este gresit: una dintre valori este negativa sau timpul minim este mai mare in comparatie cel maxim

* Aplicatia transmite actorului ca input-ul este gresit
* Scenariul se intoarce la pasul 2

1. Intervalul de timp pentru care se cere desfasurarea simularii este gresit: valoarea este negativa

* Aplicatia transmite actorului ca input-ul este gresit
* Scenariul se intoarce la pasul 2

1. Numarul de cozi este gresit: valoarea este negativa, nula sau depaseste valoarea maxima 10

* Aplicatia transmite actorului ca input-ul este gresit
* Scenariul se intoarce la pasul 2

# Proiectarea aplicatiei

Diagrama UML de clase a proiectului:

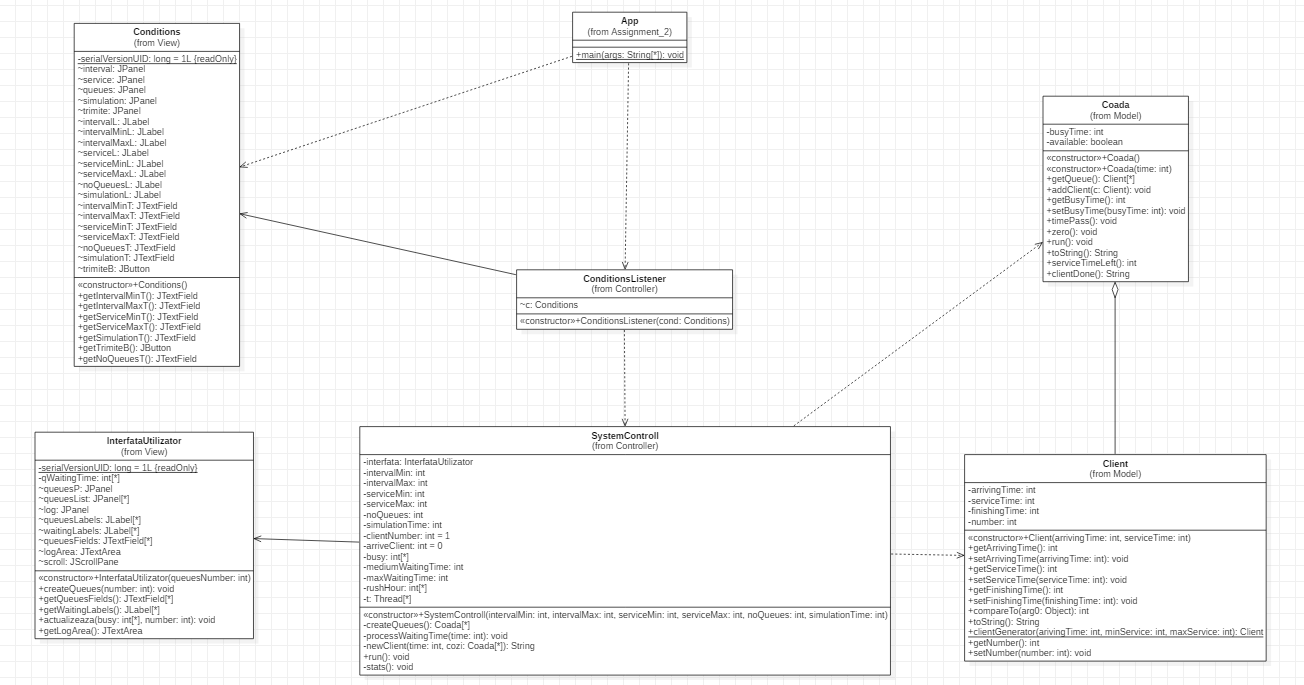
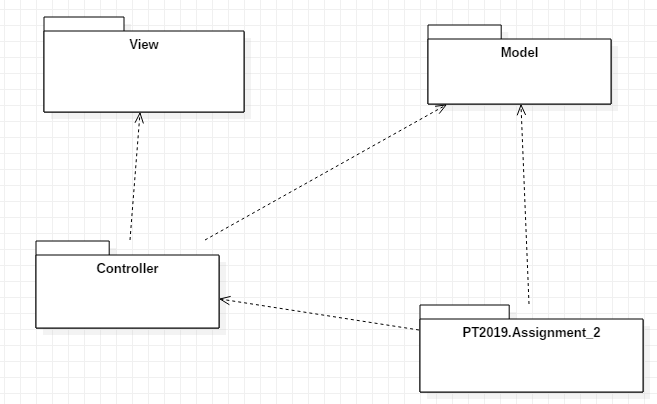


Diagrama de pachete:



Pentru o mai buna structurarea a proiectului am folosit scheletul Model-View-Controller, precurtat MVC. Astfel, clasele ce implementeaza interfata grafica se vor gasi in View, in model se afla clasele legate de modelarea aplicatiei, operatii, respectiv eventualul model matematic, pe cand controllerul realizeaza legaura intre cele doua, preluand datele de intrare introdue de catre utilizator si transmitandu-le in model unde se realizeaza operatiile. Rezultatele intoarse de catre model sunt apoi actualizate in interfata.

Interfata grafica a fost proiectata pentru a fi aerisita, focusata pe interactiunea cu utilizatorul. Astfel, rezultatul este o aplicatie cu care se interactioneaza usor si care contine titluri explicite, in acest fel fiind eliminata nevoia unui manual de utilizare.

In coada pentru a retine clientii am utilizat o lista inlantuita.

# Implementare

1. **Clasa Client – Pachetul Model**

Contine campurile :

private int arrivingTime; [\\Timpul](file:///\\Timpul) la care clientul ajunge in coada

private int serviceTime; [\\Timpul](file:///\\Timpul) necesar procesarii comenzii clientului

private int finishingTime; [\\Timpul](file:///\\Timpul) la care s-a terminat procesarea clientului

private int number; [\\Id-ul](file:///\\Id-ul) clientului

Totodata, aceasta contine pe langa metodele mutatoare si accesoare, metoda de compareTo (in urma implementarii interfetei Comparable) ce sorteaza elementele dupa arrivingTime si metoda toString, pentru o afisare a elementelor cheie ale clasei usor de interpretat de catre utilizator.

Pe langa acestea, clasa client mai contine si metoda clientGenerator care primeste ca si parametrii arriving time-ul, timpul de servire minim si timpul de servire maxim si returneaza un client cu timpul de venire primit si un timp de servire ales aleator intre minimul si maximul primit.

1. **Clasa Coada – Pachetul Model**

Contine campurile:

private LinkedList<Client> queue; [\\Reprezinta](file:///\\Reprezinta) lista de clienti ce stau la coada

private int busyTime; [\\Reprezinta](file:///\\Reprezinta) timpul necesar ca toti clientii din coada vor fi procesati

private boolean available; [\\Variabila](file:///\\Variabila) folosita in procesul de sincronizare

Totodata, aceasta contine pe langa metodele mutatoare si accesoare, metoda de run(in urma implementarii interfetei Runnable) ce contine codul pentru procesarea clientilor, managementul simularii trecerii timpului si metoda toString, pentru o afisare a elementelor cheie ale clasei usor de interpretat de catre utilizator.

Pe langa acestea, clasa coada mai contine metoda addClient ce adauga un element in queue si creste timpul busyTime cu valoarea timpului de procesare al clientului. Metoda este sincronizata pentru a se evita erorile. Tot sincronizata este si metoda accesoare al timpului de ocupare al cozii, acesta asteptand pana cand variabila available va fi facuta adevarata de catre metoda run.

Metodele timePass si zero sunt de asemenea sincronizate si sunt responsabile de atribuirea valorii „true” variabilei available, simularea trecerii timpului si respectiv anuntarea celorlalte metode ce sunt in blocate datorita apelului metodei wait(). In plus mai exista si metodele serviceTimeLeft care returneaza timpul ramas de procesare al clientilor din coada exceptandu-l pe primul si metoda clientDone, care elimina primul element din coada. Aceste doua metode au rolul de a elimina clientii dupa ce au fost procesati.

1. **Clasa conditions – Pachetul View**

Contine campurile:

JPanel interval = new JPanel();

JPanel service = new JPanel();

JPanel queues = new JPanel();

JPanel simulation = new JPanel();

JPanel trimite = new JPanel();

JLabel intervalL = new JLabel("Intervalul de sosire al clientilor:", JLabel.***CENTER***);

JLabel intervalMinL = new JLabel("Minim:");

JLabel intervalMaxL = new JLabel("Maxim:");

JLabel serviceL = new JLabel("Timpul de servire:", JLabel.***CENTER***);

JLabel serviceMinL = new JLabel("Minim:");

JLabel serviceMaxL = new JLabel("Maxim:");

JLabel noQueuesL = new JLabel("Numarul de cozi:");

JLabel simulationL = new JLabel("Timpul de simulare:");

JTextField intervalMinT = new JTextField(5);

JTextField intervalMaxT = new JTextField(5);

JTextField serviceMinT = new JTextField(5);

JTextField serviceMaxT = new JTextField(5);

JTextField noQueuesT = new JTextField(10);

JTextField simulationT = new JTextField(10);

JButton trimiteB = new JButton("Trimite");

Clasa extinde clasa Frame si contine pe langa metodele mutatoare si accesoare, constructorul care are rolul de a initializa toate valorile si de a aranja elementele din interfata. Design-ul interfetei este fabricat cu ajutorul unui GridLayout ce imparte toata suprafata in 7 linii si o singura coloana, in care se adauga apoi pe rand panourile, etichetele si campurile text corespunzatoare. Rolul acestei clase este de a prelua informatiile de initializare a simularii.

1. **Clasa InterfataUtilizator – Pachetul View**

Contine campurile:

private int[] qWaitingTime; [\\timpul](file:///\\timpul) de asteptare in fiecare coada

JPanel queuesP = new JPanel();

JPanel[] queuesList;

JPanel log = new JPanel();

JLabel[] queuesLabels;

JLabel[] waitingLabels;

JTextField[] queuesFields;

JTextArea logArea;

JScrollPane scroll;

Clasa extinde clasa Frame si contine pe langa metodele mutatoare si accesoare, constructorul care are rolul de a initializa toate valorile si de a aranja elementele din interfata. Design-ul interfetei este fabricat cu ajutorul unui GridLayout ce imparte toata suprafata in o linie si doua coloane, in care se adauga apoi pe rand panourile, etichetele si campurile text corespunzatoare cozilor, pe partea stanga si regiunea text in care se va afisa log-ul, pe partea dreapta. Rolul acestei clase este de a creea interfata grafica pe care utilizatorul sa poata urmarii in timp real desfasurarea simularii, fluxul clientilor in cozi reprezentat de timpul in care acestia intra sau parasesc o coada.

Metoda createQueues, creeaza cozile dupa numarul indicat de catre utilizator, iar metoda actualizeaza se ocupa de actualizarea labelurilor „busy for” ce indica timpul total de servire din fiecare coada la un moment de timp.

1. **ConditionsListener – Pachetul Controller**

Are rolul de a prelua conditiile initiale legate de simulare din Conditions si a le transmite mai departe pentru a se crea interfata si tot sistemul de simulare. Aceasta contine subclasa ButtonListener care implementeaza interfata ActionListener, ce are rolul de a seta un ascultator pe butonul „Trimite” din clasa Conditions. La apasarea acestui buton, se va rula metoda actionPerformed care va verifica integritatea datelor si va afisa mesaje de eroare in cazul in care acestea sunt eronate, respectiv va lansa simularea daca acestea vor fi valide.

1. **SystemControll – Pachetul Controller**

Contine campurile:

private InterfataUtilizator interfata;

private int intervalMin;

private int intervalMax;

private int serviceMin;

private int serviceMax;

private int noQueues;

private int simulationTime;

private int clientNumber = 1;

private int arriveClient = 0;

private int[] busy;

private int mediumWaitingTime;

private int maxWaitingTime;

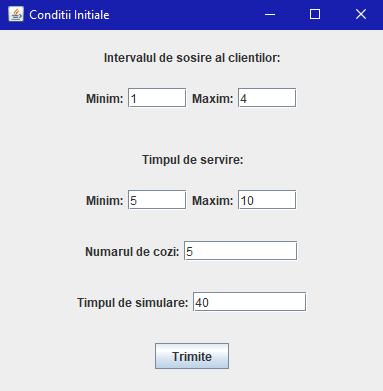
private int[] rushHour;

private Thread[] t;

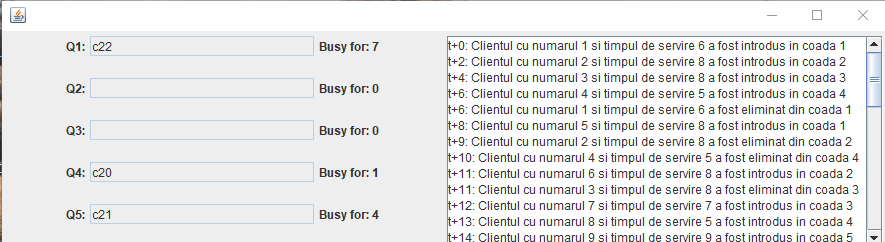
Clasa extinde interfata Runnable, implementand metoda run pe care o vom discuta ulterior. Constructorul initializeaza variabilele de clasa si creeaza interfata. Metoda createQueues creeaza cozile, le atribuie threaduri pe care apoi le porneste. Metodele processWaitingTime si stats() ajuta la calculul statisticilor si la afisarea lor. Metoda run este executata la pornirea threadului. Aceasta contine o instructiune repetetitiva care are un numar de pasi egal cu timpul de simulare introdus de catre utilizator. La fiecare iteratie, se contorizeaza numarul de clienti prezenti in coada, se updateaza log-ul daca exista evenimente, se updateaza timpul de procesare al comenzilor pentru fiecare coada si se verifica daca un client nou a ajuns la coada. In caz afirmativ se apeleaza metoda Client, care creeaza obiectul client cu timpii stabililiti, marcheaza in log aparitia sa si il returneaza in run unde se va atribui cozii cu timpul minim de asteptare.

# Rezultate

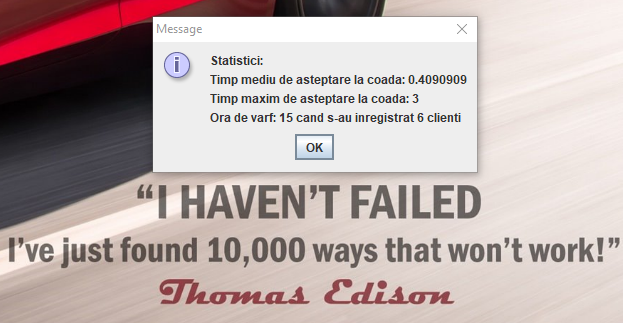
Am testat programul pentru urmatoarele valori initiale:



Cozile la finalul timpului de simulare a programului aratau in urmatoarul fel:



Statisticile dupa exectuia programului sunt:



# Concluzii

Dificultatile in dezvoltarea aplicatiei au aparut in momentul legarii partii de threaduri cu cea de interfata. Un lucru important ce a trebuit facut a fost atribuirea unui thread separat clasei SystemManager. In acest fel, cand aceasta facea request la timpii de procesare din cozi si intampina sistemul de sincronizare, putea sa astepte dupa raspunsuri fara a bloca implicit si interfata grafica.

Posibilitati de dezvoltare:

* Implementare sistem start-stop pentru a permite rularea aplicatiei pas cu pas manual
* Mai multe statistici, spre exemplu numarul de clienti in fiecare coada etc.
* Implementarea mai multor strategii de impartire a clientilor in cozi

# Referinte

<http://inf.ucv.ro/documents/tudori/laborator8_53.pdf>

<https://www.youtube.com/watch?v=VYN-CBtPNiM>