***Documentație***

***Tema 2***

***Managementul unor cozi cu***

***ajutorul thread-urilor***

Nume :Irimie Daria-Alexia, grupa 30228

# CUPRINS

[1. Obiectivul temei 3](file:///D:\Utilizator-nu%20sterge\Downloads\PT2023_Template_Documentatie_RO%20(2).doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](file:///D:\Utilizator-nu%20sterge\Downloads\PT2023_Template_Documentatie_RO%20(2).doc#_Toc95297886)

[3. Proiectare 4](file:///D:\Utilizator-nu%20sterge\Downloads\PT2023_Template_Documentatie_RO%20(2).doc#_Toc95297887)

[4. Implementare 5](file:///D:\Utilizator-nu%20sterge\Downloads\PT2023_Template_Documentatie_RO%20(2).doc#_Toc95297888)

[5. Rezultate 12](file:///D:\Utilizator-nu%20sterge\Downloads\PT2023_Template_Documentatie_RO%20(2).doc#_Toc95297889)

[6. Concluzii 13](file:///D:\Utilizator-nu%20sterge\Downloads\PT2023_Template_Documentatie_RO%20(2).doc#_Toc95297890)

[7. Bibliografie 13](file:///D:\Utilizator-nu%20sterge\Downloads\PT2023_Template_Documentatie_RO%20(2).doc#_Toc95297891)

# Obiectivul temei

**Obiectivul principal**

Aceasta tema are ca obiectiv crearea unei aplicatii care sa imite o situatie reala intalnita frecvent de fiecare dintre noi. Mai precis, aplicatia va simula o coada de asteptare similara cu cele intalnite spre exemplu in supermarket-uri si alte magazine. Aceasta aplicatie este realizata in mod eficient astfel incat clientii vor fi trimisi la cozile cele mai scurte, evitand aglomerarea. Ca prim pas, utilizatorul are rolul de a introduce datele corespunzatoare fiecarui camp prezentat in interfata, mai exact input-urile de care are nevoie aplicatia pentru a functiona.

**Obiectivele secundare**

* Analiza problemei, descrierea abordarii de implementare**-** Capitolul 2
* Proiectarea aplicatiei. Organizarea codului – Capitolul 3
* Implementarea aplicatiei de simulare a managementului cozilor – Capitolul 4
* Rezultate asteptate – Capitolul 5

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Pentru a implementa aceasta aplicatie, trebuie sa cream un sistem de management al cozilor de asteptare, care sa distribuie clientii in mod eficient intre acestea, astfel incat timpul de asteptare sa fie cat mai scurt.

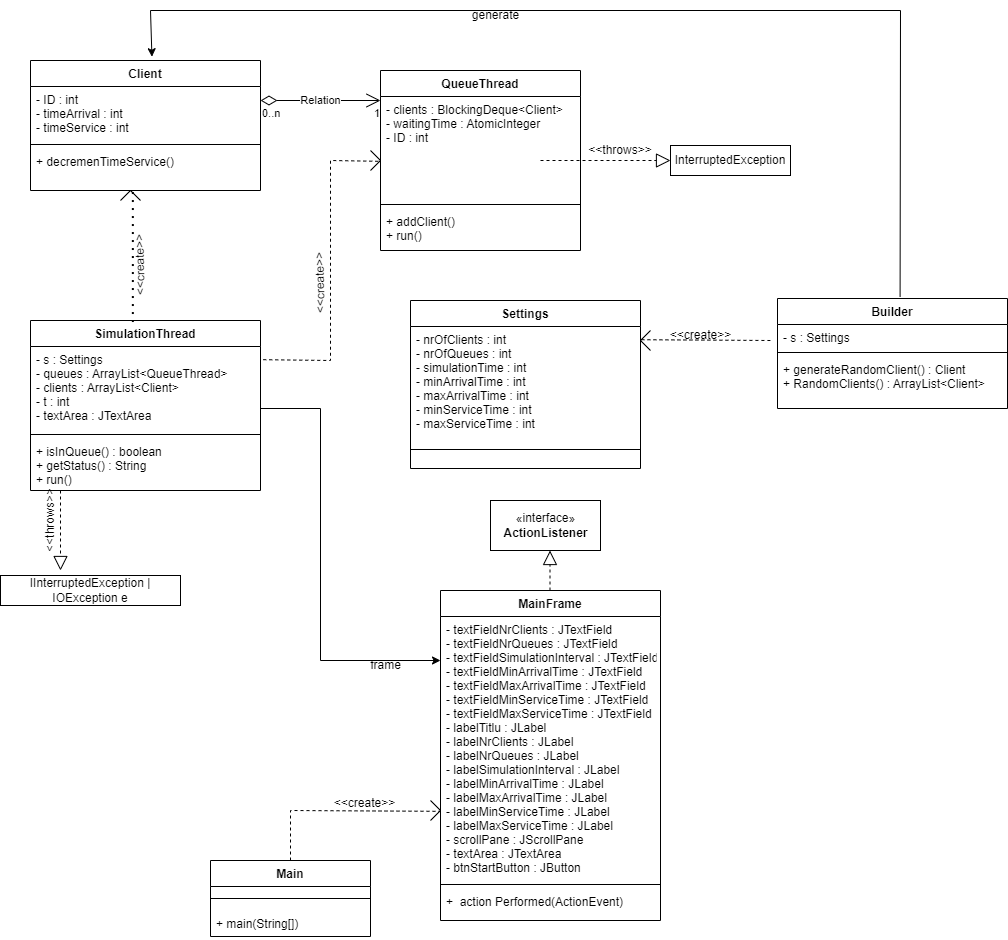
Utilizarea acestei aplicatii este simpla. Tot ce trebuie sa faca utilizatorul este sa introduca un set de date, pe baza carora se va face implementarea. Campurile acestea sunt :

* Numarul de clienti
* Numarul de cozi
* Timul de simulare
* Minimul timpului de sosire a unui client
* Maximul timpului de sosire a unui client
* Minimul timpului de servire a unui client
* Maximul timpului de servire a unui client

Dupa introducerea si citirea datelor, se apasa butonul de start, iar simularea se va porni. Utilizatorul va putea vizualiza evolutia cozilor in timp real, in fiecare secunda.

Aceste rezultate se pot observa atata in interfata grafica, cat si in fisierul .txt creat.

# Proiectare



**Organizarea programului pe clase**

* Proiectul este impartit pe clase, in functie de functionalitate :
* Campurile introduse de catre utilizator sunt reprezentate de o clasa numita Settings. Aceasta contine doar constructorul si gettere.
* Un client este reprezentat de clasa Client si are urmatoarele atribute : ID, timeArrival si timeService. ID-ul este atributul unic fiecarui client, fiind reprezentat de un numar de la 1 pana la numarul de clienti si care merge din 1 in 1. timeArrival reprezinta timpul de sosire
* al clientului. timeService reprezinta perioada de timp in care clientul va fi servit. Aceasta clasa contine constructorul, gettere si metoda toString suprascrisa. O metoda implementata in plus este decrementTimeService.
* Clientii sunt generati in mod aleator in clasa Builder, prin 2 metode. Metodele sunt generateRandomClient care returneaza un obiect de tipul Client si RandomClients care returneaza un ArrayList format din clientii generati random in metoda anterioara.
* coada este descrisa de clasa QueueThread, aceasta va contine mai multi clienti. Metoda este constituita din 3 atribute. Un BlockingDeque de tipul Client, un waitingTime de tipul Atomic Integer si un ID, int.
* Clasele QueueThread si SimulationThread contin metodele de run.

In afara acestora am implementat clasa Main care comunica cu interfata.

Graphical user interface

Description automatically generated with low confidence

# Implementare

* **Main :**

Aceasta clasa contine doar metoda main.

In metoda main se instantiaza un obiect al clasei MainFrame. Astfel, se initializeaza fereastra aplicatiei.

* **Settings :**

Campuri importante :

Aceasta clasa are ca si campuri numarul de clienti, numarul de cozi, timpul de simulare, timpul minim la care soseste un client, , timpul maxim la care soseste un client,

timpul minim de servire a unui client si timpul maxim de servire a unui client.

A picture containing graphical user interface

Description automatically generated

Metode importante :

Nu contine metode importante

* **Client :**

Campuri importante :

Graphical user interface, application

Description automatically generated

Metode importante :

Nu contine metode importante

* **Builder :**

Contine un obiect de tipul Settings

Metode importante :

* generateRandomClient : Aceasta metoda genereaza random atributele ArrivalTime si ServiceTime pentru un client. Numerele generate sunt in intervalul definit de min si max ArrivalTime si min si max ServiceTime.

Text

Description automatically generated

* RandomClients : Aceasta metoda returneaza un ArrayList de clienti, fiecare dintre ei fiind generati de catre metoda generateRandomClients. De asemenea, se trimite ca si parametru „i + 1” care va reprezenta ID-ul fiecarui client, deoarece acesta nu trebuie generat in mod aleator, avand o valoare de 1, 2, ... , nrOfClients.

Text

Description automatically generated

* **QueueThread :**

Campuri importante :

Graphical user interface, application

Description automatically generated

Metode importante :

* addClient : Aceasta metoda primeste un obiect de tipul Client si il adauga in lista de clients. De asemenea, metoda adauga la waitingTime, TimeService ul obiectului.

Graphical user interface, text

Description automatically generated

* run : Aceasta metoda ruleaza in mod continuu si se asigura ca primul client din coada este servit in ordinea sosirii, urmand ca urmatorul client sa fie preluat si servit in mod similar dupa finalizarea servirii primului client.

Functionalitatea acestei metode :

Prin clauza "try-catch" se gestioneaza situatia in care thread-ul este intrerupt din cauza unei exceptii de tipul "InterruptedException". In bucla "while(true)", se verifica constant daca coada de clienti este goală sau nu. Dacă coada este goala, programul sare peste restul instructiunilor din bucla si revine la inceputul buclei pentru a verifica din nou. Prin apelul metodei "Thread.sleep(1000)", programul face o pauza de 1 secunda inainte de a prelua primul client din coada. Prin apelul metodei "clients.element()", se preia primul client din coada. Se decrementaza timpul de servire al clientului "c" cu o unitate, folosind metoda "c.decrementTimeService()". Daca timpul de servire al clientului "c" a ajuns la 0, clientul este eliminat din coada, folosind metoda "clients.take()".

Text

Description automatically generated

* **SimulationThread :**

Campuri importante :

Graphical user interface, text

Description automatically generated

Metode importante :

* isInQueue : Metoda are rolul de a verifica daca un client se afla deja intr-una dintre cozi.

Metoda primeste ca si parametru un obiect de tipul Client, care va fi cautat in cozi. Daca timpul de servire al clientului este zero, inseamna ca clientul a fost deja servit și se returneaza valoarea "true", deoarece clientul nu mai poate fi într-o coada. Daca clientul nu a fost deja servit, se parcurg toate cozile din sistem cu ajutorul unei bucle for-each. Pentru fiecare coada, se parcurg toti clientii din acea coada cu ajutorul unei alte bucle for-each. Daca un alt client din coada respectiva este egal cu clientul dat ca parametru, inseamna ca acest client se afla deja in coada respectiva si se returneaza valoarea "true". Daca clientul nu a fost gasit in nicio coada, se returneaza valoarea "false".

Text

Description automatically generated

* isEmpty : Verifica daca coada este goala
* getStatus : Metoda returneaza un String care urmeaza sa fie afisat in Interfata Grafica si in fisierul .txt.

Se declara o variabila de tip String care va contine ceea ce dorim sa printam. Se adauga prima linie, care afiseaza mesajul „Waiting”. Se foloseste o bucla for-each pentru a parcurge toti clientii care asteapta in afara cozilor, prin apelarea metodei "isInQueue(client)". Daca un client nu este intr-o coada, se adauga informatiile despre acesta in variabila de tip String. Se afiseaza mesajul "none" daca nu exista clienti asteptand in afara cozilor. Se parcurg toate cozile din sistem, prin utilizarea unei alte bucle for-each. Pentru fiecare coada, se adauga informatii despre clientii care se afla in coada, daca exista. Daca coada este goala, se afiseaza mesajul "empty". La final, returneaza String-ul.

Text

Description automatically generated

* run : In prima parte a metodei, se incearca crearea unui fisier (outputFile). Daca fisierul nu exista, se creeaza un fișier nou. In caz contrar, se afisează un mesaj corespunzator. In continuare, se creeaza toate cozile (de tipul QueueThread) si se porneste fiecare coada intr-un fir de executie separat. Apoi, se trece la simularea efectiva. Se parcurge fiecare client si se adauga in coada daca timpul de sosire (timeArrival) este egal cu timpul curent de simulare (t). In cazul in care exista mai multe cozi cu acelasi timp minim de asteptare, clientul este adaugat la prima coada care indeplineste aceasta conditie. Dupa fiecare adaugare de client in coada, se afiseaza starea curenta a sistemului prin apelarea metodei getStatus(). Informatiile sunt stocate într-un String (finalString) si sunt afsate intr-un element de interfată grafica (textArea). In plus, informatiile sunt scrise si in fisierul de iesire (writer.println(currentStr)). Simularea continua pana cand timpul curent de simulare (t) ajunge la valoarea maxima (simulationTime). La final, se inchide fisierul de iesire (writer.close()) si se intrerup firele de executie (interrupt()). In cazul in care apare o exceptie (InterruptedException sau IOException), se afiseaza un mesaj corespunzator.
* **MainFrame :**

Aceasta clasa se ocupa de partea grafica a programului. Reprezinta interfata vizuala si logica din spatele programului.

Campuri importante :

7 textField-uri, un buton de start si JtextArea pentru a afisa rezultatele simularii :

Graphical user interface, text, application, chat or text message

Description automatically generated

Aceasta clasa implementeaza ActionListener pentru butonul de start, care porneste simularea.

In interiorul acestuia, se preiau valorile introduse de utilizator in campurile de text pentru numarul de clienti, numarul de cozi, intervalul de simulare, timpul minim si maxim de sosire si timpul minim si maxim de servire, folosind metoda "Integer.parseInt()". Aceste valori sunt folosite pentru a crea un obiect de tip "Settings" (configurari) și un obiect de tip "Builder" care genereaza o lista aleatorie de clienti folosind aceste configurari. Pentru fiecare client din lista de clienti generata, se afisează informatiile despre client folosind metoda "System.out.println(client.toString())". Se creeaza un obiect de tip "SimulationThread" care primeste ca argumente obiectul "Settings", lista de cliensi si un obiect "TextArea" pentru afisarea mesajelor in timpul simularii. Se porneste thread-ul simularii folosind metoda "simThr.start()".

Text

Description automatically generated

# Rezultate

Rezultatele simularii se pot observa in Interfata Grafica.

Application

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generated

# Concluzii

Prin intermediul acestei aplicatii, utilizatorul poate simula si gestiona eficient o coada de asteptare, primind informatii in timp real. La sfarsitul simularii, utilizatorul poate sa vizualizeze informatiile in aplicatie sau intr-un fisier text. De asemenea, pentru a lucra cu thread-uri, aplicatia utilizeaza metode si structuri de date special adaptate.

Aplicatia ar putea fi imbunatatita. De exemplu, poate fi implementata o alta strategie de distribuire a clientilor la cozile create.

# Bibliografie

<https://www.w3schools.com/java/java_threads.asp>

<https://www.w3schools.com/java/java_files_create.asp>

<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html>

<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingDeque.html>