DOCUMENTATIE

TEMA *2*

NUME STUDENT: Dragus Andreea

GRUPA: 30226

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 3](#_Toc95297887)

[4. Implementare 3](#_Toc95297888)

[5. Rezultate 3](#_Toc95297889)

[6. Concluzii 3](#_Toc95297890)

[7. Bibliografie 3](#_Toc95297891)

# Obiectivul temei

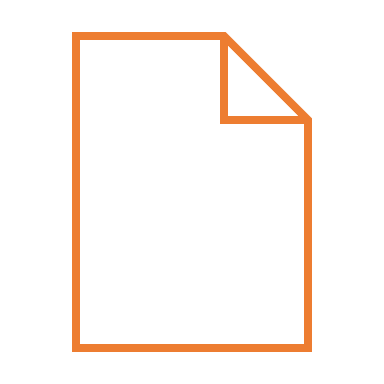
Obiectivul principal al temei este reprezentat de simularea unui scenariu real, in care clientii (unui magazin de exemplu) se aseaza la mai multe cozi si asteapta sa fie serviti. Acest lucru este realizat prin generarea aleatoare a unor clienti (denumite task-uri in cod) si a unor cozi (implementate cu ajutorul thread-urilor). Pe baza unui criteriu (care poate fi ales in functie de lungimea cozii sau de timpul de asteptare la o coada), in momentul in care clientii ajung la “magazin” acestia sunt redirectionati catre cea mai favorabila coada din perspectiva criteriului ales. Acest procedeu se repeta pe durata timpului alocat simularii.

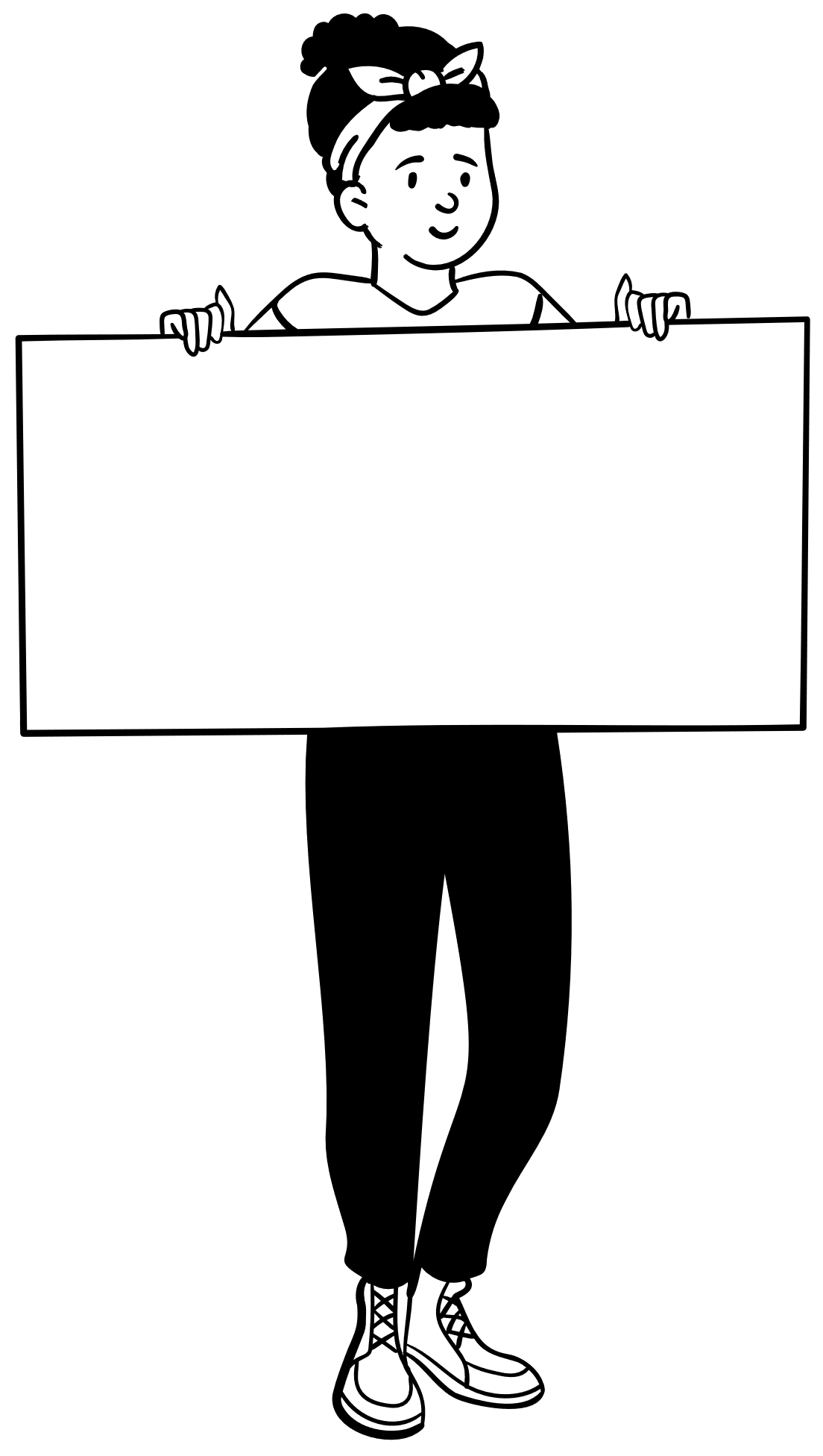
# Analiza problemei, modelare, scenarii, cazuri de utilizare

Utilizatorul interactioneaza cu proiectul propriu-zis prin intermediul GUI. In cadrul acesteia, utilizatorul introduce datele conform carora va fi modelata simularea:

* numarul clientilor
* numarul cozilor
* intervalul de simulare (timpul maxim de simulare): simularea incepe la indicatorul de timp 0 cand sunt generati clientii si se incheie la momentul de timp egal cu input-ul primit de la utilizator cand se inchid toate cozile
* minimul si maximul timpului la care clientii pot sa ajunga
* minimul si maximul timpului necesar pentru a servi un client

Clientii vor fi generati aleator, avand fiecare din ei un id, un timp la care ajung la magazin ( generat aleator in intervalul introdus de utilizator) si un timp de servire ( generat aleator in intervalul introdus de utilizator).





A picture containing text, screenshot

Description automatically generated

USER

# Proiectare

Acest proiect este impartit in 3 pachete, intrucat respecta modelul arhitectural MVC. Contine 8 clase si o interfata.

Pachetul „controller” contine clasele ButtonListener si SimulationManager. Pachetul „model” contine clasele Scheduler, Server, ShortestStrategy, TimeStrategy, Task si interfata Strategy. Pachetul „view” contine clasa GUI.

Interfata utilizator este alcatuita din mai multe textFiled-uri in cadrul carora utilizatorul poate introduce elementele mentionate mai sus. De asemenea, aceasta contine si un buton intitulat „START SIMULATION” care sa initieze intregul proces de simulare a cozilor si clientilor. Dupa aceste elemente, se afla o sectiune intitulata „text area” in cadrul careia se vor afisa in timp real evenimentele care au loc la fiecare indicator de timp. Simularea se termina in momentul in care indicatorul temporal a ajuns la o valoare egala cu cea introdusa de utilizator in sectiunea „Simulation interval” sau in momentul in care toti clientii generati au fost serviti.

Diagrama UML:

# 

# Implementare

1. Pachetul „model”
   1. Task:

* Este clasa care reprezinta clientii. Aceasta are ca si variabile instanta mai multe campuri private de tip int: id, arrivalTime, serviceTime si un camp privat static de tip int: index. ArrivalTime reprezinta timpul la care clientul ajunge la magazin, serviceTime reprezinta cat dureaza sa fie servit iar campul id ia valoarea variabilei statice index care se incrementeaza de fiecare data cand se creeaza un obiect de tipul Task.
* Aceasta clasa contine un constructor in cadrul caruia sunt initializate variabilele instanta mentionate mai sus si incrementata variabila statica id.
* De asemenea, aceasta clasa implementeaza interfata Comparable cu ajutorul careia se vor ordona clientii in ordinea timpului de sosire.
* Clasa Task contine 6 metode:

1. public String toString(): returneaza un string cu componentele unui obiect de tip Task.
2. public int getArrivalTime(): metoda de get pentru campul private arrivalTime
3. public void setArrivalTime(int arrivalTime): metoda de set pentru campul private arrivalTime
4. public int getServiceTime(): metoda de get pentru campul private serviceTime
5. public void setServiceTime(int serviceTime): metoda de set pentru campul private serviceTime
6. public int compareTo(Task o): compara obiectul cu referinta this cu un alt obiect de tipul Task pentru a obtine o ordine crescatoare a timpului de sosire
   1. Server:

* Este clasa care reprezinta cozile. Aceasta are mai multe variabile instanta:

1. BlockingQueue<Task> tasks: este o structura de tip coada care retine task-urile care se afla la un momentdat in cadrul unui server (fie ca sunt in desfasurare fie ca asteapta sa fie preluate). Tipul de date folosit este BlockingQueue intrucat dorim sa ne asiguram ca nu vom intampina situatii in care 2 threaduri sa modifice aceasta coada de task-uri simultan
2. AtomicInteger waitingPeriod: calculeaza timpul de asteptare la o coada. Tipul de date folosit este din nou unul care sa blocheze threadurile din a modifica valoarea acestui camp simultan
3. AtomicInteger avgWaitingPeriod, avgServiceTime, avgTaskNumber, avgTaskServiceNumber: sunt variabile instanta care vor fi folosite pentru calcularea timpului de asteptare mediu si a timpului de servire mediu
4. serverId si index au acelasi rol ca si in clasa Task;
5. boolean open are rolul de a marca daca o coada este deschisa (daca mai sunt task-uri in ea) sau nu.

* Aceasta clasa are un constructor care primeste ca si parametru un numar intreg, reprezentand numarul maxim de task-uri care se pot afla in coada tasks. In cadrul acestui constructor, se initializeaza variabilele instanta tasks, serverId si open.
* De asemenea, aceasta implementeaza interfetele Runnable si Comparable
* Clasa Server contine 20 de metode, dintre care cele mai importante sunt:

1. public void run(): aceasta metoda este executata de catre thread-ului asociat serverului. In cadrul unei bucle vom verifica daca mai exista task-uri in server, iar daca da, atunci dorim sa extragem primul task din coada. Daca serviceTime-ul acestuia este mai mare ca 0, atunci il vom decrementa si pe acesta si waitingPeriod-ul serverului, iar daca este egal cu 0 atunci il vom scoate din coada.
2. public void addTask(Task newTask): aceasta metoda are rolul de a adauga un nou task in coada de task-uri a serverului. De asemenea, aceasta actualizeaza waitingPeriod-ul serverului dar si datele necesare calcularii mediilor mentionate mai sus.
3. public float getAverageWaitingPeriod(): returneaza perioada medie de asteptare aferenta server-ului prin impartirea perioadei de asteptare la numarul de task-uri introduse in coada
4. public float getAverageServiceTime(): returneaza timpul mediu de servire aferent server-ului prin impartirea timpului de servire la numarul de task-uri introduse in coada
5. public String toString(): returneaza sub formele unui string datele despre coada de task-uri aferenta serverului
6. public int compareTo(Server o): compara obiectul cu referinta this cu un alt obiect de tipul Server in functie de waitingPeriod-ul lor pentru a determina care din acestia are cel mai mic waiting period, prin urmare este cel in care se va introduce task-ul la timpul curent
7. Celelate metode care nu au fost descrise mai sus sunt metode de get si set pentru variabilele instanta ale clasei.
   1. Strategy:

* Este o interfata care contine definitia metodei addTask(ArrayList<Server> serverList, Task t).
  1. TimeStrategy:
* Aceasta clasa implementeaza interfeata Strategy
* In metoda addTask: ordoneaza ArrayList-ul de servere primit ca parametru in functie de waitingPeriod, atsfel incat pe prima pozitie sa fie serverul cu cel mai mic timp de asteptare. Dupa aceea, task-ul este adaugat la serverul de pe pozitia 0 din array prin intermediul metodei addTask din clasa Server descrisa mai sus.
  1. ShortestStrategy:
* Aceasta clasa implementeaza interfata Strategy
* In metode addTask: vom cauta serverul care are lista de task-uri cu size-ul cel mai mic si la aceasta vom adauga task-ul primit ca si parametru
  1. Scheduler:
* Aceasta clasa are ca si variabile instanta un array list de servere, un array list de threaduri care va fi asociat fiecarui server, un obiect de tip Strategy prin intermediul caruia se va selecta una din cele doua strategii definite mai sus si doi intregi care definesc numarul maxim de servere respectiv numarul maxim de task-uri care pot fi alocate unui server.
* Clasa Scheduler contine un constructor prin intermediul caruia se initializeaza campurile maxServersNumber si maxTasksPerServerNumber. De asemenea, sunt creeate arrayList-urile de servere si threaduri; atunci cand se creeaza un server care va fi adaugat la arrayList-ul servers, se creeaza si un thread asociat acestui server care va fi adaugat in arrayList-ul serversThreads.
* Aceasta clasa contine 14 metode, dintre care cele mai importante sunt:

1. public void dispatchTask(Task task): aceasta metoda instantiaza un obiect din tipul dorit de strategie, in cazul de fata TimeStrategy. In continuare, cu obiectul creat se va apela metoda addTask care va distribui task-ul primit ca si parametru serverului care are cel mai mic timp de asteptare
2. public String toString(): are rolul de a retuna un string care sa contina informatii despre toate serverele aflate in array list-ul servers.
3. public void stop(): are rolul de a seta campul „open” al tuturor serverelor drept „false”
4. Metodele care nu au fost mentionate aici sunt metode de get si set pentru variabilele instanta ale clasei.
5. Pachetul „controller”:
   1. SimulationManager:

* Aceasta este clasa care pune cap la cap toate elementele enuntate pana acum si da viata simularii.
* Aceasta are mai multe variabile instanta: numarul de clienti, intervalul de siimulare, timpul minim de ajungere, timp maxim de ajungere, timp minim servire, timp maxim servire, textArea in care se vor afisa informatiilor, ArrayList<Task> task care reprezinta task-urile generate, un obiect de tip scheduler, 4 variabile de tip boolean cu ajutorul carora putem sa selectam dintre 4 moduri: mod normal cu afisare in textArea, si 3 moduri de test in cadrul carora se vor afisa datele in fisierele text file1, file2, file3. Pentru scrierea in fisier s-au folosit 3 bufferedWriter-uri cu ajutorul carora informatiile sa fie atasate la finalul fisierelor si nu suprascrise. De asemenea, avem si variabilele avgWaitingTime, avgServiceTime si peakHour pentru calcularea informatiilor statistice de medie.
* SimulationManager are un constructor in cadrul caruia sunt initializate variabilele instanta si sunt creeate fisierele text in care se va scrie in cazul in care sunt selectatele modurile de test. De asemenea, sunt initializate fileWriter-urile si bufferedWriter-urile cu ajutorul carora se va realiza scrierea in fisiere. Tot in cadrul constructorului, este apelata metoda generate care are rolul de a genera task-urile (numarul lor este variabila instanta numberOfClients) cu atribute temporale aflate in intervalele definite de utilizator. Aceste task-uri sunt memorate in ArrayList-ul „tasks”. De asemenea, este instantiat obiectul de tip scheduler si sunt pornite thread-urile din variabila instanta serverThreads a scheduler.
* Clasa in discutie are 2 metode, extrem de importante:

public void generate(): aceasta metoda are rolul de a genera un arrayList de task-uri care au arrivalTime si serviceTime cuprins in intervalele specificate de utilizator in GUI. De asemenea, arrayList-ul generat este ordonat crescator in functie de arrivalTime.

public void run(): In cadrul unei bucle verificam daca variabila time (care porneste de la 0) este mai mica decat simulationInterval introdus de utilizator si mai avem task-uri in array-ul de task-uri generat sau chiar daca array-ul de task-uri este gol, mai avem servere deschise care inca servesc clienti. Daca in array-ul de task-uri avem task-uri care au arrivalTime-ul egal cu timpul curent, atunci dorim sa le trimitem scheduler-ului care prin intermediul metodei dispatchTask o va asigna serverului cu timpul cel mai mic de asteptare. De asemenea, task-ul va fi extras din array-ul de task-uri intrucat a fost deja asignat unui server. Se incepe construirea string-ului rezultat care contine la fiecare pas informatii despre momentul de timp la care ne aflam, arrayList-ul de task-uri generate si servere. Verificand in care dintre cele 4 moduri ne aflam, vom afisa in textArea informatiile sau le vom scrie in unul din fisiere. Pentru calculul peakHour am parcurs lista de servere a scheduler-ului si am verificat la fiecare moment de timp size-ul acestora, afland astfel timpul simularii in care am avut in servere cele mai multe task-uri concomitent. Pentru calcului timpului mediu de asteptare si servire, am adunat intr-o variabila timpurile medii de asteptare/servire a tuturor serverelor si la final am impartit rezultatele la numarul de servere. La final, am inchis fisierele deshise si am oprit simularea prin inchiderea tuturor serverelor.

* 1. ButtonListener:
* Aceasta clasa are ca si variabile instanta: private JTextField nrClients, private JTextField nrQueues, private JTextField simulationTime, private JTextField arrivalMin, private JTextField arrivalMax, private JTextField serviceMin, private JTextField serviceMax,  
  private JTextArea textArea;
* Clasa ButtonListener are un constructor prin intermediul caruia sunt initializate variabilele instanta mentionate mai sus.
* In metoda public void actionPerformed(ActionEvent e) se preiau datele introduse de catre utilizator in GUI in text-fielurile variabila instanta prin metoda Integer.ParseInt(); dupa aceea se instantiaza un obiect de tipul SimulationManager care primeste ca si parametrii ai constructorului informatiile obtinute de la GUI si textArea-ul in care vor fi scrise detaliile simularii. De asemenea, este creeat si pornit thread-ul principal al proiectului, asociat obiectului simulationManager. Astfel, in momentul in care utilizatorul apasa butonul „start simulation” sa inceapa derularea evenimentelor.

1. Pachetul „view”
   1. GUI: contine implementarea interfetei grafice
   2. Main: contine metoda main

# Rezultate

Au fost verificate 3 scenarii de test ale caror rezultate au fost scrise in fisiere text.

# 

# Concluzii

Aceasta tema poate fi dezvoltata prin implementarea ei pe nevoile unui magazin real pentru eficientizarea interactiunii cu clientii.

# Bibliografie

Informatiile necesare implementarii acestui proiect au fost preluate din prezentarea pusa la dispozitie si geeksforgeeks.