**DOCUMENTATIE**

**TEMA 2**

**Nume student: Mihali Gabriel**

**Grupa: 30225**

Contents

[**1.** **Obiectivul temei:** 3](#_Toc99479730)

[**2.** **Analiza problemei, modelare, scenarii, cazuri de utilizare** 4](#_Toc99479731)

[**3.** **Proiectare** 4](#_Toc99479732)

[**4.** **Implementare** 5](#_Toc99479733)

[5. **Concluzii:** 10](#_Toc99479734)

[**6.** **Bibliografie:** 11](#_Toc99479735)

# **Obiectivul temei:**

* **Obiectivul principal** : Implementarea unei aplicatii, cu interfata grafica pentru management-ul unor cozi, care atribuie clientilor cozi astfel incat timpul de asteptare sa fie minimizat.
* **Obiectivele secundare :** 
  + **Crearea unei clase client:** Clasa Client va avea setter-e, getter-e si parametri care ajuta la indeplinirea obiectivului principal, ca si timpul de sosire, timpul de servire, id. Mai multe detalii despre aceasta clasa in capitolul 4.
  + **Crearea interfetei pentru input-ul simularii** **:** Interfata va fi implementata folosind Java Swing, intr-un mod placut vizual si inteligibil. Mai multe detalii despre implementarea interfetelor in capitolul 4.
  + **Crearea interfetei pentru simularea live :** Interfata va fi implementata folosind Java Swing, cu actualizare la fiecare secunda, masurata de thread-ul de management. Mai multe detealii despre implementarea interfetelor in capitolul 4.
  + **Crearea unui generator de client random :** Folosind un algoritm cu modulo si functia rand(), acesta genereaza n clienti random in functie de niste parametri dati, care urmeaza sa fie introdusi in simulare si in cozi. Algoritmul din spatele generarii este explicat in capitolele 3 si 4.
  + **Crearea unor cozi de client sub forma unor thread-uri :** Crearea unei clase de tip QueueThread, cu o colectie de clienti thread-safe ca si BlockingQueue, si parametri thread-safe ca si AtomicInteger. Structurile folosite pentru cozile de client si thread sunt explicate in capitolul 3.
  + S**incronizarea thread-urilor :** Folosind functia sleep(), thread-urile coada sunt sincronizate, in functie de service time-ul clientilor. De asemenea sincronizarea este obtinuta prin metodele de tip synchronized, prin campurile thread-safe, ca si AtomicInteger si colectii thread-safe ca si BlockingQueue. Metodele de sincronizare a thread-urilor folosite sunt explicate in capitolul 3.

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

* **Proiectati si implementati** o aplicatie de gestiune a cozilor care atribuie clienti cozilor astfel incat timpul de asteptare sa fie minimizat. Obiectivul principal al unei cozi este de a oferi un loc de asteptare pentru un client inainte de a primi un serviciu. Sistemele de management pentru cozi sunt folosite pentru a minimiza acest timp de asteptare pentru fiecare client, inainte de a fi servit, in functie de anumite criterii, ca si timpul minim de asteptare si numarul minim de client din fata.
* **Date de input:** 
  + **N** clienti
  + **Q** cozi
  + **t\_maxsimulation** , timpul maxim pentru simulare
  + **minimumarrivaltime,** timpul maxim pentru sosire
  + **maximumarrivaltime,** timpul minim pentru sosire
  + **minimumservicetime,** timpul minim pentru servire
  + **maximumservicetime,** timpul maxim pentru servire

Folosind aceste date de input, aplicatia genereaza N client random cu parametri intre **minimumarrivaltime** si **maximumarrivaltime, minimumservicetime** si **maximumservicetime**. Aplicatia porneste Q thread-uri si lanseaza simularea cozilor, pana **la t\_maxsimulation,** atribuind clienti in cozi in functie de strategia de selectie.

# **Proiectare**

* Aplicatia are 9 clase, un enum si un interface, distribuite in 2 pachete, Model si View. Pachetul View are 2 clase, SimulationView si LiveLogView. SimulationView se ocupa de input-ul utilizatorului si de setarea parametrilor pentru simulare, iar LiveLogView se ocupa de simularea live si afisarea valorilor simularii.
* In pachetul Model, avem 7 clase, un enum si o interfata. Interfata definita este Strategy, care descrie o metoda addClient, care urmeaza sa fie implementata in ConcreteStrategyTime si ConcreteStrategyQueue in functie de strategia de selectie. ConcreteStrategyTime foloseste interfata Strategy, metoda addClient este implementata astfel : se adauga clientul curent in coada cu cel mai mic timp de servire total. ConcreteStrategyQueue foloseste interfata Strategy, metoda addClient fiind implementata astfel : se adauga clientul current in coada cu cei mai putini clienti. Clasa Client modeleaza clientul cu 3 parametri, Clasa ManageThread foloseste Runnable si reprezinta thread-ul care se ocupa de generarea clientilor si de timpul de simulare. La fiecare 1000ms, acesta apeleaza functia sleep() si numara secundele. ObjectComparator este o clasa care foloseste Comparator pentru a sorta listele de clienti in functie de un anumit criteriu si anume, timpul de sosire. QueueThread este o clasa care foloseste Runnable. Aceasta reprezinta un thread care tine o coada de clienti. Clasa Scheduler creeaza thread-urile pentru cozile de clienti si schimba strategia de selectie in functie de input-ul utilizatorului, adaugand clienti in diferite cozi. Aceasta clasa este data ca si parametru in ManageThread. SelectionPolicy este un enum care contine cele 2 tipuri de strategii de selectie : SHORTEST\_QUEUE si SHORTEST\_TIME.
* Ca si structuri de date speciale, lucrul cu thread-uri implica structuri thread-safe pentru a nu fi accesate concurent de mai multe thread-uri. Astfel avem AtomicInteger si BlockingQueue, structuri care sunt implicit thread-safe si au metode speciale, ca si addAndGet si element().
* Algoritmul de generare a clientilor din clasa ManageThread foloseste clasa Random si metoda nextInt() din aceasta clasa pentru a genera un int random. Pentru a seta anumite limite parametrice se foloseste modulo (% x) pentru a genera numere intre 0 si x – 1. Formula folosita este dupa modelul urmatoarei :

**((rand % (max – min)) + (min + 1))** – genereaza un numar random intre min si max.

# **Implementare**

* Aplicatia are 9 clase, un enum si un interface, distribuite in 2 pachete, Model si View.
  + **Pachetul Model: Se ocupa de partea de implementare a aplicatiei.**
    - **Clasa Client :**  Clasa Client reprezinta clientul propriu-zis, care este bagat sau scos din coada. In constructor, ca si parametri primeste un id, un timp de sosire si un timp de servire, generate random. Contine 6 getter-e si 6 setter-e si un toString(), pentru afisarea detaliilor despre Client.
    - **Clasa ManageThread:** Foloseste ca si interfata Runnable, avand metoda run(), pentru o functionalitate de tip thread. Are ca si campuri, numarul de clienti, numarul de cozi, numarul maxim de secunde pentru simulare, politica de selectie, timpul de servire minim, timpul de servire maxim, timpul de sosire minim si timpul de sosire maxim. Acestia sunt parametrii pentru simulare introdusi de catre utilizator. De asemenea, aceasta clasa contine un camp de tip Scheduler, si o lista synchronized care contine clientii generati random de catre metoda generateRandomClients(), explicata in capitolul 3. Metoda run() descrie comportamentul thread-ului. Acesta incrementeaza variabila currentTime la fiecare secunda, thread-ul apeland metoda sleep() la fiecare 1000ms. Cu un foreach, lista de client generati este parcursa, iar clientii care au timpul de sosire egal cu timpul simularii sunt introdusi intr-o coada, in functie de politica de selectie. De asemenea clasa ManageThread mai contine si metoda main, care porneste aplicatia

.

* + - **Clasa QueueThread :** Ca si clasa ManageThread, foloseste ca si interfata Runnable, pentru a fi folosita intr-un thread. Ca si campuri, are o structura de tip BlockingQueue de clienti care reprezinta clientii care stau la coada la secunda respectiva, perioada totala de asteptare in toata coada, un AtomicInteger, numarul maxim de clienti, numele cozii, de tip String, si logView, care este interfata in care aceasta clasa scrie datele live. Ca si metode, avem metoda run(), luata de la Runnable, care descrie comportamentul Thread-ului QueueThread. Acesta selecteaza primul client din coada folosind metoda element(), asteapta o perioada de service time al unui client, folosind metoda sleep() si il sterge din coada, afisand aceasta actiune de stergere in interfata live. QueueThread are o metoda de addClient, care adauga un client in coada si adauga service time-ul acestui client in waitingPeriod, folosind metoda addandGet, specifica AtomicInteger-ului. Aceasta clasa are un constructor care primeste ca parametru view-ul de simulare, numele si numarul maxim de client. Are 3 getter-e pentru nume, lista de client si perioada totala de asteptare : getClients(), getWaitingPeriod(), getName().
    - **Clasa Scheduler :** Are 5 campuri,queues, lista de cozi, numarul maxim de cozi, numarul maxim de clienti, strategia de selectie, si view-ul live. Are un constructor care primeste ca parametru view-ul live, numarul maxim de cozi si numar maxim de clienti. In acest constructor Scheduler creeaza cozile de client de tip QueueThread, si porneste thread-urile fiecarei cozi cu metoda start(). Are un getter, getQueues(), care returneaza lista de cozi. Metoda changeStrategy() selecteaza strategiile de selectie ConcreteStrategyQueue si ConcreteStrategyTime in functie de SelectionPolicy dat ca si parametru din interfata utilizatorului. Metoda dispatchTask() apeleaza addClient(), si adauga clientul dat intr-o coada in functie de strategia de selectie, formata mai sus, in metoda changeStrategy().
    - **Enum SelectionPolicy:** Contine numele strategiilor de selectare, SHORTEST\_QUEUE si SHORTEST\_TIME. Acestea definesc tipurile de selectie a cozilor. Prima , SHORTEST\_QUEUE, adauga clienti in cozi in functie de lungimea cozilor. A doua, SHORTEST\_TIME, adauga clienti in cozi in functie de timpul de asteptare total al cozii.
    - **Interface Strategy:** Strategy este o interfata care defineste metoda addClient, ulterior implementata in ConcreteStrategyQueue si ConcreteStrategyTime.
    - **Clasa ObjectComparator:** O clasa care foloseste metodele interfetei Comparator, implementeaza metoda compare si compara doua clase de tip Client, in functie de timpul de sosire. Se foloseste pentru ordonarea crescatoare a listelor de client in functie de timpul de sosire.
    - **Clasa ConcreteStrategyQueue:** Foloseste interfata Strategy si implementeaza metoda addClient() dupa strategia SHORTEST\_QUEUE. Aceasta metoda primeste ca parametru o structura de tip List de QueueThread, cozi cu clienti si gaseste Queue-ul cu cel mai mic numar de clienti, astfel: parcurge lista de QueueThread-uri, gaseste cea mai mica coada prin folosirea metodei size(), salveaza marimea cozii intr-o variabila si gaseste coada egala cu size-ul variabilei printr-un foreach.
    - **Clasa Concrete StrategyTime:** Foloseste interfata Strategy si implementeaza metoda addClient() dupa strategia SHORTEST\_TIME. Aceasta metoda primeste ca parametru o structura de tip List de QueueThread, cozi cu client si gaseste Queue-ul cu cel mai mic timp total de asteptare. Numarul total de asteptare este suma tuturor timpilor de service ale tuturor clientilor aflati intr-un anumit moment intr-o coada. Metoda parcurge lista de QueueThread-uri, gaseste cel mai mic timp total de asteptare prin metoda getWaitingPeriod(), salveaza timpul de asteptare intr-o variabila si gaseste coada care are timpul de asteptare minim printr-un foreach.
  + **Pachetul View : Se ocupa de partea de interfata aplicatiei.**
    - **Clasa SimulationView:** Este clasa in care utilizatorul introduce input-ul si porneste simularea de pe butonul Start. Mai jos este prezentata o imagine cu interfata SimulationView. Timpii minimi si maximi de sosire si de servire sunt introdusi in niste JTextField-uri. Numarul maxim de secunde ale simularii si numarul de cozi sunt introduce in JTextField-uri. Interfata contine un JButton de start, caruia ii este atribuit un ActionListener care deschide interfata de simulare live. Tipul de selectie pentru adaugarea in cozi este selectat dintr-un JComboBox cu optiunile SHORTEST\_QUEUE si SHORTEST\_TIME. Interfata are 6 getter-e pentru timpul minim si maxim de sosire, timpul minim si maxim de servire, numarul de cozi si numarul de secunde pentru simulare.

**In figura de mai jos este prezentata diagrama pachetelor:**

**A picture containing text, whiteboard

Description automatically generated**

Aceasta diagrama ilustreaza pachetele aplicatiei, clasele pe care le contin si relatiile dintre ele. Model importa pachetul View, clasele LiveLogView si SimulationView. Clasele ConcreteStrategyQueue, ConcreteStrategyTime, QueueThread si Scheduler importa clasa LiveLogView iar clasa ManageThread importa ambele clase din View. Din diagrama se observa ca toate clasele sunt de tip public.

**In imaginea de mai jos este prezentata diagrama UML a pachetului Model.**

**Diagram

Description automatically generated with medium confidence**

Aceasta diagrama arata campurile fiecarei clase, tipurile campurilor, modificatorii de acces a fiecarui camp. De asemenea diagrama UML prezinta metodele fiecarei clase si relatiile dintre aceste clase. In diagrama avem mai multe interfete : Comparator, Strategy, Runnable. In relatie cu aceste interfete sunt clasele ConcreteStrategyQueue, ConcreteStrategyTime, QueueThread, ObjectComparator si ManageThread. Clasa Scheduler contine o lista de clase QueueThread, si foloseste interfata Strategy.

**Graphical user interface, application

Description automatically generatedIn figura de mai jos este prezentata interfata SimulationView.**

* + - **Clasa LiveLogView:** Este clasa care afiseaza output-ul simularii aplicatiei. Contine un JTextArea care afiseaza mesaje despre adaugari si stergeri din cozi, un log cu actiunile pe care le simuleaza aplicatia. Acest JTextArea este adaugat intr-un JScrollPane. Contine 2 JTextField-uri care afiseaza secundele curente si perioada medie de asteptare a tuturor clientilor. Spre deosebire de SimulationView, aceasta clasa are setter-e : setSeconds() pentru a afisa secundele curente, setAverage() pentru a afisa timpul de asteptare mediu a tuturor clientilor si addTextArea() pentru a adauga text despre actiunile aplicatiei in JTextArea, respectiv JScrollPane. Aceste metode sunt de tip synchronized, setarea textului in interfata facandu-se intr-un mod care asigura siguranta thread-urilor.

**In figura de mai jos este prezentata interfata LiveLogView.**

**Graphical user interface, application, Word

Description automatically generated**

# **Concluzii:**

* Una dintre cele mai utile aplicatii pentru algoritmi, pentru programare in general, este eficientizarea cozilor. O problema reala, care, rezolvata, poate aduce foarte multe beneficii.
* Aceasta aplicatie a rezultat la dezvoltarea cunostintelor despre thread-uri, despre procese si despre notiunile de sincronizare a thread-urilor, de multithreading in general. Au fost puse in lumina modurile de utilizare ale thread-urilor si motivele pentru care se folosesc. Notiunile despre Java Swing au fost aprofundate, notiunile despre JScrollPane si despre JPanel. Au fost asimiliate cunostinte noi despre sortarea colectiilor despre interfata Comparator, despre sortarea dupa anumite criterii, despre interfete si implementarea metodelor interfetelor.
* Ca si dezvoltari ulterioare, poate fi dezvoltata interfata grafica a utilizatorului, pot fi adaugate animatii, pentru o intelegere mai usoara a simularii.

# **Bibliografie:**

https://dsrl.eu/courses/pt/

https://www.educative.io/edpresso/how-to-generate-random-numbers-in-java

https://www.baeldung.com/java-synchronized-collections

https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html