**DOCUMENTATIE TEMA 2**

**MANAGEMENT DE COZI FOLOSIND THREAD-URI SI MECANISME DE SINCRONIZARE**

**Rad Daniel-Cristian**

**Grupa 30223**

Contents

[1.Obiective 3](#_Toc164344684)

[1.1.Obiectiv principal 3](#_Toc164344685)

[1.2.Obiective secundare 3](#_Toc164344686)

[2.Analiza problemei 3](#_Toc164344687)

[2.1.Clase 3](#_Toc164344688)

[2.2.Scenarii de utilizare/use case-uri 4](#_Toc164344689)

[3.Proiectare 4](#_Toc164344690)

[3.1. Diagrama UML 4](#_Toc164344691)

[3.2. Structuri de date 6](#_Toc164344692)

[3.3. Interfete 6](#_Toc164344693)

[3.4. Algoritmi utilizati 7](#_Toc164344694)

[4.Implementare 7](#_Toc164344695)

[4.1. Clasa Client 7](#_Toc164344696)

[4.2. Clasa Server 7](#_Toc164344697)

[4.3. Clasa Generator 7](#_Toc164344698)

[4.4. Clasa Scheduler 7](#_Toc164344699)

[4.5. Interfata Strategy 7](#_Toc164344700)

[4.6. Clasa Simulator 8](#_Toc164344701)

[4.7. Clasa Logger 8](#_Toc164344702)

[4.8. Clasa InputForm 8](#_Toc164344703)

[5. Rezultate 8](#_Toc164344704)

## 1.Obiective

### 1.1.Obiectiv principal

Obiectivul principal al temei este de a creea o aplicatie, mai exact o simulare a unui sistem de management al unor cozi, folosind thread-uri si metode de sincronizare a threadurilor cu scopul de a obtine timpul de asteptare si de procesare a sarcinilor minim.

### 1.2.Obiective secundare

Aceste obiective secundare ne ajuta treptat sa ajungem la atingerea obiectivului principal. Cateva dintre obiective noastre secundare ar putea fi urmatoarele:

* Generarea clientilor in mod aleatoriu. Creearea de clienti cu un ID, un timp de sosire si un timp de procesare aleatoriu.
* Alocarea clientilor anumitor cozi, in functie de anumite criterii. Se vor folosi 2 strategii pentru plasarea clientilor in cozi: shortest time si shortest queue.
* Gestionarea timpului de asteptare si procesare a clientilor . Se va monitoriza timpul de asteptare si timpul de procesare a clientilor pentru o buna evaluare a strategiilor de alocare in cozi.
* Afisarea rezultatelor simularii. Se vor prezenta rezultatele simularii in interfata grafica si intr-un fisier logger.

## 2.Analiza problemei

In cadrul acestei teme, am analizat problema distribuirii clientilor intre mai multi serveri, avand in vedere obtinerea timpului minim de asteptare si de procesare al sarcinilor. Pentru realizarea acestui lucru, vom modela si defini urmatoarele clase si use case-uri:

### 2.1.Clase

* **Client:** reprezinta clientul propriu-zis cu un id, un timp de sosire si un timp de procesare.
* **Server:** reprezinta un server care proceseaza clientii. Acesta este format dintr-o coada de clienti si un timp de asteptare pentru clientii alocati.
* **Generator:** este folosit pentru generarea aleatoare a clientilor.
* **Scheduler:** acesta este folosit pentru a plasa clientii la serverii corespunzatori astfel obtinandu-se timpul minim de astepare si de procesare.
* **Simulator:** acesta se ocupa de simularea propriu-zisa.
* **Strategy:** este o interfata folosita pentru a selecta strategia dorita in plasarea clientilor la serveri.

### 2.2.Scenarii de utilizare/use case-uri

* Caz 1: Generarea clientilor

Generatorul genereaza clienti cu un id, un timp de sosire si un timp de procesare aleatoare. Acestia sunt transmisi mai departe pentru a fi procesati.

* Caz 2: Alocarea clientilor la serveri

Scheduler-ul aloca clientul la un server in functie de strategia selectata.

* Caz 3: Procesarea clientilor de catre serveri

Serverii proceseaza clientii in ordinea in care au fost alocati. Timpul de procesare al fiecarui client este monitorizat si actualizat corespunzator.

* Caz 4: Afisarea rezultatelor simularii

Rezultatele simularii sunt afisate pentru a evalua eficienta strategiilor de alocare a clientiilor in cozi.

## 3.Proiectare

In proiectarea acestei aplicatii am folosit principiile OOP si am creeat mai multe clase care comunica intre ele pentru rezolvarea sarcinilor. In continuare se vor prezenta structurile de date, interfetele, clasele si algoritmii utilizati in proiectarea aplicatiei.

### 3.1. Diagrama UML

A screenshot of a computer screen

Description automatically generated

Clasele din aplicatie sunt organizate in urmatoarele pachete:

* Gui: pachetul unde sunt organizate elementele care apartin interfetei grafice
* Logic: pachetul unde sunt organizate elementele care apartin logicii aplicatiei
* Model: pachetul unde sunt organizate clasele Server si Client

Client: Clasa Client reprezinta un client care trebuie procesat de unul dintre serveri. Aceasta clasa contine campuri pentru stocarea identificatorului unic al clientului, timpul de sosire și timpul de servire al acestuia. De asemenea, include metode pentru a obtine și seta aceste valori, precum si o metoda toString() pentru a afisa informatii despre client intr-un format usor de citit.

Server: Clasa Server reprezinta un server care gestioneaza o coada de sarcini. Aceasta contine o coada blocanta (BlockingQueue) pentru a stoca clientii și un AtomicInteger pentru a gestiona timpul total de asteptare pentru clientii din coada. Clasa Server implementeaza Runnable pentru a putea rula intr-un fir de executie separat, permitand simularea simultana a mai multor serveri. Metodele importante din această clasă includ addClient() și run().

Scheduler: Clasa Scheduler este responsabila pentru gestionarea serverilor si atribuirea clientilor la acestia folosind o strategie specifica. Aceasta contine o listă de serveri și o instanta a interfetei Strategy. Scheduler permite schimbarea strategiei in functie de politica de selectie dorita și gestioneaza adaugarea clientilor la serveri prin intermediul metodei dispatchClient().

Strategy: Interfata Strategy defineste metoda addClient() care trebuie implementata de clasele TimeStrategy și ShortestQueueStrategy. Această metoda primeste o lista de serveri si un client si atribuie clientul unui server conform strategiei implementate.

ShortestQueueStrategy: Clasa ShortestQueueStrategy implementeaza interfata Strategy si strategia de atribuire a clientilor la serverul cu cea mai scurta coada. În metoda addClient(), aceasta gaseste serverul cu cea mai scurta coada și adauga clientul la acesta.

TimeStrategy: Clasa TimeStrategy implementeaza interfata Strategy si strategia de atribuire a clientilor la serverul cu cel mai scurt timp de asteptare. In metoda addClient(), aceasta gaseste serverul cu cel mai scurt timp de asteptare si adauga clientul la acesta.

### 3.2. Structuri de date

In aceasta aplicatie am folosit 2 structuri de date: un BlockingQueue in clasa Server pentru a putea stoca clientii si o lista in clasa Scheduler pentru a putea gestiona serverii disponibili.

public class Scheduler {  
 private List<Server> servers;

}

public class Server implements Runnable {  
 private BlockingQueue<Client> clients

}

### 3.3. Interfete

In aceasta aplicatie am folosit o interfata Strategy cu o metoda addClient(). Aceasta interfata este implementata de cele 2 strategii pentru a putea adauga clientii la serveri in diferite moduri.

public interface Strategy {  
 void addClient(List<Server> servers, Client client);  
}

### 3.4. Algoritmi utilizati

Algoritmii utilizati sunt urmatorii:

* TimeStrategy, unde cautam serverul cu cel mai mic timp de asteptare si alocam clientul acestuia
* ShortestQueueStrategy, unde cautam serverul cu cei mai putini clienti in coada si adaugam clientul la acesta.

## 4.Implementare

In aceasta parte vom descrie implementarea claselor, metodelor importanet si intrerfata utilizator. Se vor prezenta, de asemenea, modul in care clasele aplicatiei comunica intre ele.

### 4.1. Clasa Client

Clasa Client contine trei campuri pentru a stoca informatiile despre ID-ul clientului, timpul de sosire si timpul de servire. Clasa include, de asemenea, metode pentru a obtine aceste valori si o metoda toString().

### 4.2. Clasa Server

Clasa Server implementeaza interfata Runnable si contine o coada de tipul BlockingQueue pentru a stoca clientii. De asemenea, include un camp de tipul AtomicInteger pentru a gestiona perioada de asteptare a serverului. Clasa contine metode pentru adaugarea clientilor si obtinerea timpui de asteptare.

### 4.3. Clasa Generator

Clasa Generator se ocupa de generarea aleatoare a clientilor care vor primi un id, un timp de sosire si un timp de servire random cuprinse intr-un interval. Aceasta clasa transmite mai departe lista de clienti generata catre Simulator.

### 4.4. Clasa Scheduler

Clasa Scheduler gestioneaza o lista de serveri si se ocupa de plasarea clientilor utilizant o strategie specificata. Clasa contine metode pentru initializarea serverilor, schimbarea strategiei si pentru expedierea clientilor catre serveri.

### 4.5. Interfata Strategy

Interfata Strategy defineste o metoda addClient() pentru adaugarea clientilor la serveri. Clasele TimeStrategy si ShortestQueueStrategy implementeaza aceasta interfata si folosesc algoritmi diferiti pentru expedierea clientilor catre serveri.

### 4.6. Clasa Simulator

Clasa Simulator se ocupa cu simularea propriu-zisa. Acesta contine instante ale claselor generator si scheduler si incepe simularea de la un timp 1 pana la un timp maxim setat din interfata de utilizator. Acesta mai are o referinta si catre o clasa Logger care se ocupa cu logarea informatiilor referitoare la simulare.

### 4.7. Clasa Logger

Clasa Logger se ocupa de logarea informatiilor referitoare la simulare si le scrie intr-un fisier .txt. Acesta ne arata evolutia la fiecare timp al simularii a cozilor si a clientilor care asteapta sa fie plasati.

### 4.8. Clasa InputForm

Clasa InputForm reprezinta interfata grafica a aplicatiei. Aceasta a fost realizata cu Swing. In cadrul acestei interfete se vor introduce de la tastatura datele de intrare necesare pornirii simularii.

## 5. Rezultate

În această secțiune, vom prezenta scenariile de testare și rezultatele obținute în urma rulării aplicației. Scenariile de testare vor ilustra funcționalitățile aplicației și vor demonstra eficiența diferitelor strategii de alocare a sarcinilor.