DOCUMENTATIE

TEMA 1

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

NUME STUDENT: NEGRUT CIPRIAN

GRUPA: 302110

**1. Obiectivul temei**

Obiectivul principal al temei este realizarea unei aplicatii de management de cozi intr-un magazi, utilizand thread-uri si sincronizari.

Obiectivele secundare sunt urmatoarele:

<> Implementarea clasei [ Main ]

-utilizata pentru rularea aplicatiei

<> Implementarea clasei [ GUI\_APP ]

-interfata utilizator, interactiunea utilizatorului cu aplicatia

<> Implementarea clasei [ ConcreteStrategyQueue ]

-strategia pentru Shortest\_Queue

<> Implementarea clasei [ ConcreteStrategyTime ]

-strategia pentru Shortest\_Time

<> Implementarea clasei [ Controller ]

-utilizata pentru a lega datele introduse de catre utilizator si logica din spatele aplicatiei, pentru a genera un raspus dorit

<> Implementarea clasei [ Scheduler ]

-initializarea unui [ Server ] cu o lista de [ Task-uri ]

<> Implementarea clasei [ Server ]

-reprezinta cozile din magazin unde se aseaza clientii

<> Implementarea clase [ SimulationManager ]

-rularea thread-urilor pentru fiecare [ Server ]

<> Implementarea clasei [ Task ]

-reprezinta un client

**2. Analiza problemei, modelare, scenarii, cazuri de utilizare**

Cadrul de cerinte functionale cuprinde: SelectionPolicy.SHORTEST\_QUEUE si SelectionPolicy.SHORTEST\_TIME. Acestea ne ajuta sa adaugam un client intr-o coada cu cei mai putini clienti sau intr-o coada cu cel mai putin timp de asteptare.

Use-case-ul: SHORTEST\_QUEUE

<> Actorul princial: User-ul

<> Scenariul de succes principal:

1. Utilizatorul introduce in TexField-uri valorile pentru numarul de cozi, numarul de clienti, simulationTime-ul si intervalul de sosire a clientului la coada si intervalul de servire a unui client

2. Utilizatorul apasa pe butonul [ SHORTEST\_QUEUE ] pentru a selecta criteriul dupa care se face adaugarea in coada a fiecarui client

3. Utilizatorul apasa pe butonul [ Start ] si incepe simularea

Use-case-ul: SHORTEST\_TIME

<> Actorul princial: User-ul

<> Scenariul de succes principal:

1. Utilizatorul introduce in TexField-uri valorile pentru numarul de cozi, numarul de clienti, simulationTime-ul si intervalul de sosire a clientului la coada si intervalul de servire a unui client

2. Utilizatorul apasa pe butonul [ SHORTEST\_TIME] pentru a selecta criteriul dupa care se face adaugarea in coada a fiecarui client

3. Utilizatorul apasa pe butonul [ Start ] si incepe simularea

Use-case-ul: RESET

<> Actorul principal: User-ul

<> Scenariul de succes principal:

1. Utilizatorul introduce in TexField-uri valorile pentru numarul de cozi, numarul de clienti, simulationTime-ul si intervalul de sosire a clientului la coada si intervalul de servire a unui client

2. Utilizatorul apasa butonul [ RESET ]

3. Dupa apasarea butonuli [ RESET ] de catre utilizator, simularea se opreste instant, iar din scorllbar-urile de pe ecran se sterge toata informatia

4. Valorile introduse initial de catre User raman in textField-urile respective

**3. Proiectare**

Clasa Main:

<> Prin clasa Main se ruleaza aplicatia

Clasa GUI\_APP:

<> Interfata pentru utilizator, contine 7 TextField-uri pentru setarea valorilor si a intervalelor pentru a porni o simulare corecta

<> Implementeaza ActionListener pentru fiecare buton asociat fiecarei operatii

<> Contine butoanele pentru START, RESET, SHORTEST\_QUEUE si SHORTEST\_TIME

Clasa Controller:

<> Realizeaza o legatura intre datele introduse de catre utilizator si logica aplicatiei

Clasa ConcreteStrategyQueue:

<> Are implementata logica pentru a sorta cozile astfel incat prima coada de pe prima pozitie sa fie coada cu cel mai mic size

Clasa ConcreteStrategyTime:

<> Are implementata logica pentru a sorta cozile astfel incat prima coada de pe prima pozitie sa fie coada cu cel mai mic tim de asteptare

Clasa Scheduler:

<> Initializeaza un [ Server ] cu un numar de clienti si creeaza cate un thread pentru fiecare client

Clasa Server:

<> Contine o lista de clienti si implementeaza o metoda de run pentru thread-uri

Clasa Task:

<> Contine toate detaliile pe care un client trebuie sa le aiba

Clasa SimulationManager:

<> Reprezinta logica de simulare a thread-urilor pentru toate serverele

**4. Implementare**

Clasa ConcreteStrategyQueue:

<> Metode:

- addTask: primeste ca si parametrii o lista de Server si un Task. Ordoneaza lista de servere in functie de timpul minim de astepatare pe fiecare Server. Adauga in un Task pe prima pozitie in lista de servers.

Clasa ConcreteStrategyTime:

<> Metode:

- addTask: primeste ca si parametrii o lista de Server si un Task. Ordoneaza lista de servere in functie de size-ul minim de pe fiecare Server. Adauga in un Task pe prima pozitie in lista de servers.

Clasa Controller:

<> Constructori:

- Controller: primeste ca si parametru un view, creeaza obiectul de tipul PrinterWriter pentru a crea fisierul “printFileLogs.txt”

<> Campuri:

- view: putem accesa metodele din clasa [ GUI\_APP ]

- simManager: putem accesa metodele din clasa [ SimulationManager ]

- policy: ia valoarea [ SelectionPolicy ] setata in urma apasarii butoanelor SHORTEST\_QUEUE sau SHORTEST\_TIME

- obj: este folosit pentru creearea unui fisier “printFileLogs.txt”

<> Metode:

- updateView: afiseaza pe ecran detaliile in urma rularii thread-urilor

- closeFileLog: inchide fisierul “printFileLogs.txt”

- actionPerformed: ia decizii in functie de ce buton se apasa de pe fereastra GUI\_APP

~ la apasarea butonului [ START ] se iau toate informatiile din TextField-uri si se trimit ca si parametrii in constructorul SimulationManager

~ la apasarea butonului [ SHORTEST\_QUEUE ] se seteaza tipul strategiei de adaugarea a unui client in coada

~ la apasarea butonului [ SHORTEST\_TIME ] se seteaza tipul strategiei de adaugarea a unui client in coada

~ la apasarea butonului [ RESET ] se reseteaza scrollbar-urile din view, si se opreste instant rularea simularii

Clasa Scheduler:

<> Constructori:

- Scheduler: primeste ca si parametrii numarul maxim de cozi si numarul maxim de clienti dintr-o coada. Adauga in lista de servere maxTasksPerServer clienti

<> Campuri:

- servers: o lista de servere

- maxNoServer: numarul maxim de cozi

- maxTasksPerServer: numarul maxim de clienti pe care il poate avea o coada

- strategy: strategia dupa care se baga fiecare client in coada

<> Metode:

- changeStrategy: creeaza un obiect nou de timpul strategiei selectate in urma apasarii butoanelor din view

- dispatchTask: adauga in coada un client in functie de strategia aleasa

Clasa Server:

<> Campuri:

- tasks: reprezinta lista de clienti dintr-un server

- waitingPeriod: numarul de asteptare pana la servirea unui client

- running: ia valoarea true atat timp cant server.isEmpty() e false

<> Metode:

- addTask: adauga un client in coada si incrementeaza waitingPeriod cu timpul de servire a clientului adaugat recent in coada

- run: ruleaza thread-urile din clasa [ Server ]

- compareTo: compara doua servere in functie de waitingPeriod

- compareToSize: compara doua servere in functie de tasks.size()

Clasa Task:

<> Constructori:

- Task: initializeaza ID, arrivalTime si serviceTime cu valorile primite ca si parametrii

<> Campuri:

- ID: id-ul clientului

- arrivalTime: timpul de sosire pentru fiecare client

- serviceTime: timpul de servire pentru fiecare client

<> Metode

- getters and setters pentru fiecare camp si o metoda toString pentru afisarea id-ului, arrivalTime si seriveTime

Clasa SimulationManager:

<> Constructori:

- SimulationManager:

~ initializeaza numberOfClients, numberOfServers, timeLimit, minArrivalTime, maxArrivalTime, minProcessingTime, maxProcessingTime, controller cu valorile primite ca si parametrii

~ initializeaza campul scheduler cu un obiect de timpul Scheduler(numberOfservers, numberOfClients)

~ seteaza policy-ul pentru strategy

~ apeleaza functia de generare a N radom tasks

<> Campuri:

- controller: putem accesa metode din clasa [ Controller ]

- currentTime: timpul curent

- timeLimit: limita de timp a simularii

- maxProcessingTime: timpul de servire maxim

- minProcessingTime: timpul de servire minim

- minArrivalTime: timpul de sosire minim

- maxArrivalTime: timpul de sosire maxim

- numberOfServers: numarul de cozi

- numberOfClients: numarul de clienti intr-o coada

- running: are valoarea true pana cand cozile de clienti nu sunt goale sau butonul de resete nu este apasat

- scheduler: putem accesa metode din clasa [ Scheduler ]

- generatedTasks: o coada de clienti generati random

<> Metode:

- generateNRandomTasks: genereaza un numberOfClients de clienti random si returneaza o lista de clienti generati random

- checkForRun: verifica daca cozile de clienti nu sunt goale

- run: ruleaza thread-urile pentru SimulationManager

**5. Rezultate**

Rezultate pentru:

<> Test1:

- N = 4 Q = 2 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 60 seconds [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ,𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [2, 30] [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [2, 4]



<> Test2:

- N = 50 Q = 5 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 60 seconds [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ,𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [2, 40] [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [1, 7]



<> Test3:

- N = 1000 Q = 20 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 200 seconds [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ,𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [10, 100] [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [3, 9]



**6. Concluzii**

In concluzie, de-a lungul implementarii acestei aplicatii, m-am lovit de unele lucruri pe care nu le stiam in java, dar cu putin research am reusit sa le rezolv si sa si invat in acelasi timp lucruri noi. Aceasta tema m-a ajutat sa inteleg mai bine cum se folosesc thread-urile.

**7. Bibliografie**

Site-urile folosite: <https://www.geeksforgeeks.org/>