Universitatea Tehnică din Cluj-Napoca

Facultatea de Automatizări și Calculatoare

Student : Stanciu Ioan-Octavian

An: 2

Grupa: 30223

Profesor: Dorin Vasile Moldovan

**TEHNICI DE PROGRAMARE FUNDAMENTALE**

**TEMA 2**

**SIMULARE DE COZI**

**CUPRINS:**

* Obiectivul temei
* Analiza problemei, modelare, scenarii, cazuri de utilizare
* Proiectare
* Implementare
* Rezultate
* Concluzii
* Bibliografie

***1) Obiectivul temei***

Obiectivul acestei teme este de crea o aplicatie care simuleaza un sistem de "punere" a clientilor la coada cu timpul cel mai scurt de asteptare

Obiective secundare:

* Afisare in fisier a simularii de tipul
* Timpul la care se afla simularea ( in secunde )
* Clientii care asteapta sa fie asignati unei cozi ( Waiting clients : (id, arrivalTime, serviceTime)
* Cozile cu id-ul lor si clientii care sunt asignati la timpul t0
* Generarea random a N clienti si verificarea daca arrivalTime-ul si serviceTime-ul sunt cuprinse in intervalul specificat de input
* Crearea unei clase de tip Compartor pentru a sorta clientii generati in functie de arrivalTime
* Implementarea interfetei utilizator folosing Model-View-Controller design pattern

Toate aceste obiective vor fi detaliate in capitolul 4.Implementare

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

In aceasta problema se poate simula in interfata grafica aceasta problema cu maxim 4 cozi, in timp ce in consola se poate simula cu cate cozi se doreste:

Descriere use-case pentru simulare in GUI:

* Se va deschide programul
* Se schimba calea absoluta a fisierului Results.txt din FileWriter din clasa SimulationManager din metoda run. Rezultatele se vor afisa si in fisier. ATENTIE! Daca in fisier este deja ceva scris se va scrie de la inceputul fisierului
* Se va rula metoda main din clasa Controller
* Se vor introduce in interfata grafica urmatoarele:
* Timpul minim si maxim in care un client poate sa fie asignat unei cozi (minProcessingTime si maxProcessingTime)
* Timpul minim si maxim in care un client sa fie servit cand ajunge primul la coada (minServiceTime si maxServiceTime)
* Numarul de cozi care se doreste a fi folosite (Queues - maximul este de 4)
* Numarul de clienti ce se doreste a fi generati random (Clients)
* Timpul in secunde in care se doreste sa se intample simulare (daca se introduc N secunde atunci timul o sa fie in intervalul [0, N-1]
* Se apasa pe butonul "Enter Input" pentru a incepe simularea
* Rezultatele o sa se afiseze si in fisierul Results.txt (daca se respecta a doua bulina prezentata mai sus)

***3) Proiectare***

Diagrama UML:

Graphical user interface

Description automatically generated with low confidence

* Prima decizie pe care am luat-o a fost sa ma ajut de prezentarea oferita de domnul profesor.
* Dupa am implementat clasele si metodele prezentate acolo
* Am realizat ca de fapt clasa Task din prezentare este clasa Client din proiect si clasa Server este clasa Queue
* Am adaugat in clasa Client si ID-ul ca variabila instanta pentru a putea vizualiza mai usor rezultatele dupa simulari
* In clasa Queue am completat metoda run exact dupa indicatiile din prezentare si mi-am mai creat o metoda de getWaitingPeriod care imi returneaza watingPeriod-ul cozii de tip AtomicInteger
* Dupa mi-am scris corpul metodelor addTask din clasele ConcreteStrategyQueue si ConcreteStrategyTime de adaugare a unui client intr-o coada unde coada este cea mai scurta respectiv cea cu perioada de asteptare cea mai mica
* Dupa in clasa Scheduler mi-am construit constructorul si am implementat toate metodele ce au ramas
* Iar in clasa SimulationManager mi-am construit constructorul si mi-am scris metoda run si mi-am mai creat o metoda numita getClientsInQueue
* Am simulat de asemenea input-urile date in prezentarea temei si memorate in fisierele Input1.txt pentru primul input, Input2.txt pentru al doilea input si Input3.txt pentru al treilea input
* Iar la final am implementat interfata utilizator folosindu-ma de Model-View-Controller design pattern

***4) Implementare***

* Clasa Client:

1) Ca si variabile instanta am 3 toate fiind de tip int si reprezinta ID-ul clientului (ID), momentul in care acesta poate sa fie asignat unei cozi (arrivalTime) si timpul care arata cat dureaza serviceTime (processingTime)

2) In constructor imi initializez toate aceste 3 variabile instanta cu 3 variabile transmise ca parametrii

3) Urmatoarele 6 metode reprezinta gettere si settere pentru acestea 3

* Clasa Queue:

1) Aceasta clasa implementeaza interfata Runnable

2) Ca variabile instanta am 2, prima fiind de tip BlockingQueue<Client> unde stochez toti clientii asignati unei anumite cozi (clients) si a doua este de tip AtomicInteger pentru a incrementa, respectiv decrementa perioada de asteptare (waitingPeriod)

3) In constructor imi initializez prima variabila cu un LinkedBlockingQueue de dimensiune maxima de 1000, iar a doua variabila cu 0

4) In medota addClient imi adaug in coada clientul transmis ca si parametru si dupa incrementez perioada de asteptare cu processingTime-ul acestuia

5) In metoda run parcurg lista de clienti si pun mai intai Thread-ul pe sleep pentru atatea secunde cat este nevoie ca primul client sa fie servit, dupa care decrementez waitingPeriod-ul cozii cu processingTime-ul clientului si dupa il scot din coada. Toate acestea trebuie facute in interiorul unui block de tip try-catch datorita Thread-ului

6) Dupa am 2 getere si 2 setere pentru variabilele instanta

* Interfata Strategy:

1) In aceasta interfata este o singura metoda pentru a adauga in client la o coada (in functie de timpul de asteptare sau de numarul de clienti deja asignati)

* Clasa ConcreteStrategyQueue:

1) Aceasta clasa implementeaza interfata Strategy si adauga un client in coada in functie de numarul de clienti deja asignati.

2) Parcurg lista de cozi si daca o coada are mai putini clienti decat urmatoarea atunci retin minte pozitia primei cozi si pozitia celei de a doua in caz contrar.

* Clasa ConcreteStrategyTime:

1) Ca si clasa ConcreteStrategyQueue, clasa ConcreteStrategyTime implementeaza interfata Strategy si asigneaza un client la o coada in functie de timpul minim de asteptare.

2) Imi instantiez un obiect de tip Queue si ii setez waitingPeriod-ul la valoarea 1000000. Imi iau un contor pentru a retine pozitia cozii cu waitingPeriod-ul minim si de asemenea in obiectul instantiat retin minimul de waitingPeriod

3) La final imi adaug in coada de pe pozitia poz clientul t

* Clasa SortClients:

1) Aceasta clasa am creat-o pentru a putea sorta crescator toti clientii in functie de timpul la care acestia ajung si de aceea aceasta clasa implementeaza clasa Comparator<Client>

* Enum SelectionPolicy:

1) Acest enum are rolul de a selecta ce politica vream sa folosim pentru asignarea clientilor in cozi (dupa timp sau dupa lungimea cozii)

* Clasa Scheduler:

1) Am 4 variabile instanta prima fiind de tip List<Queue> (queues) unde o sa fie stocate toate cozile generate. Este initializata cu un ArrayList<>() sincronizata. Urmatoarele 2 variabile instanta sunt de tip int si reprezinta numarul maxim de cozi (maxNoQueues) si numarul maxim de clienti per coada (maxClientsPerQueue). Ultima variabila instanta este de tip Strategy (strategy) si este initializata cu un obiect de tip ConcreteStrategyTime().

2) In constructor imi generez maxNoQueues cozi si le adaug in queues, iar pentru fiecare coada generata in creez un thread ce are ca obiect ultima coada generata (la fiecare thread se apeleaza .start())

3) Metoda changeStrategy are rolul de a schimba strategia in functie de cum se doreste. Schimbi in ConcreteStrategyQueue daca vrei stocarea clientilor in cozi dupa dimensiunea minim a lor sau in ConcreteStrategyTime daca vrei stocarea clientilor in cozi dupa cel mai scurt timp.

4) Metoda dispatchClient are rolul de a apela metoda de addTask din strategy adaugand clientul t transmis ca parametru in antetul metodei

5) Ultima metoda este un getter pentru a lua toata lista de cozi

* Clasa SimulationManager:

1) In aceasta clasa am 13 variabile instanta: primele 7 sunt de tip int si reprezinta timpul total in care trebuie sa se intample simularea (timeLimit), timpul minim si maxim in care un client poate sa fie asignat unei cozi (minProcessingTime, maxProcessingTime), timpul minim si maxim care dureaza pentru un client sa fie servit (minServiceTime, maxServiceTime), numarul total de cozi folosite (numberOfServers) si numarul total de clienti ce vor fi asignati (sau nu) cozilor (numberOfClients).

Urmatoare variabila este de tip View din MVC pentru a putea sustrage datele introduse de utilizator in interfata grafica. Urmatoarele 3 variabile sunt timpul mediu de servire (averageServiceTime), timpul mediu de asteptare (averageWaitingTime) si secunda la care au fost cei mai multi asignati. Cea de a 12-a variabila instanta este de tip Scheduler (scheduler), iar ultima este de tip lista unde se vor stoca cei numberOfClients clienti generati random ( generatedClients)

2) In constructor imi initializez view-ul si primele 7 variabile instanta urmate de averateWaitingTime pe care il impart initial la numarul de clienti. Variabila instanta scheduler o initializez cu numarul de cozi si numarul de clienti dupa care apelez metoda de generare a numberOfClients clienti pe care ii stochez in generatedClients

3) In metoda generateNRandomClients imi generez N clienti. Pentru fiecare client imi generez random un processTime si un arrivalTime in intervalul [1, maxProcessTime] respectiv [1,maxArrivalTime]. Dupa asta verific daca cele doua au fost generate mai mici decat minProcessTime respectiv maxProcessTime, iar daca da atunci se aduna minimul. Daca toate au fost generate bune atunci se adauga in generatedClients noul client cu id-ul i+1 (i ul fiind contorul din bucla for). Dupa finalizarea buclei se sorteaza clientii dupa in functie de arrivalTime cu un obiect de tip SortClients. La final se calculeaza averageServiceTime care este media aritmetica a tuturor processingTime a clientilor.

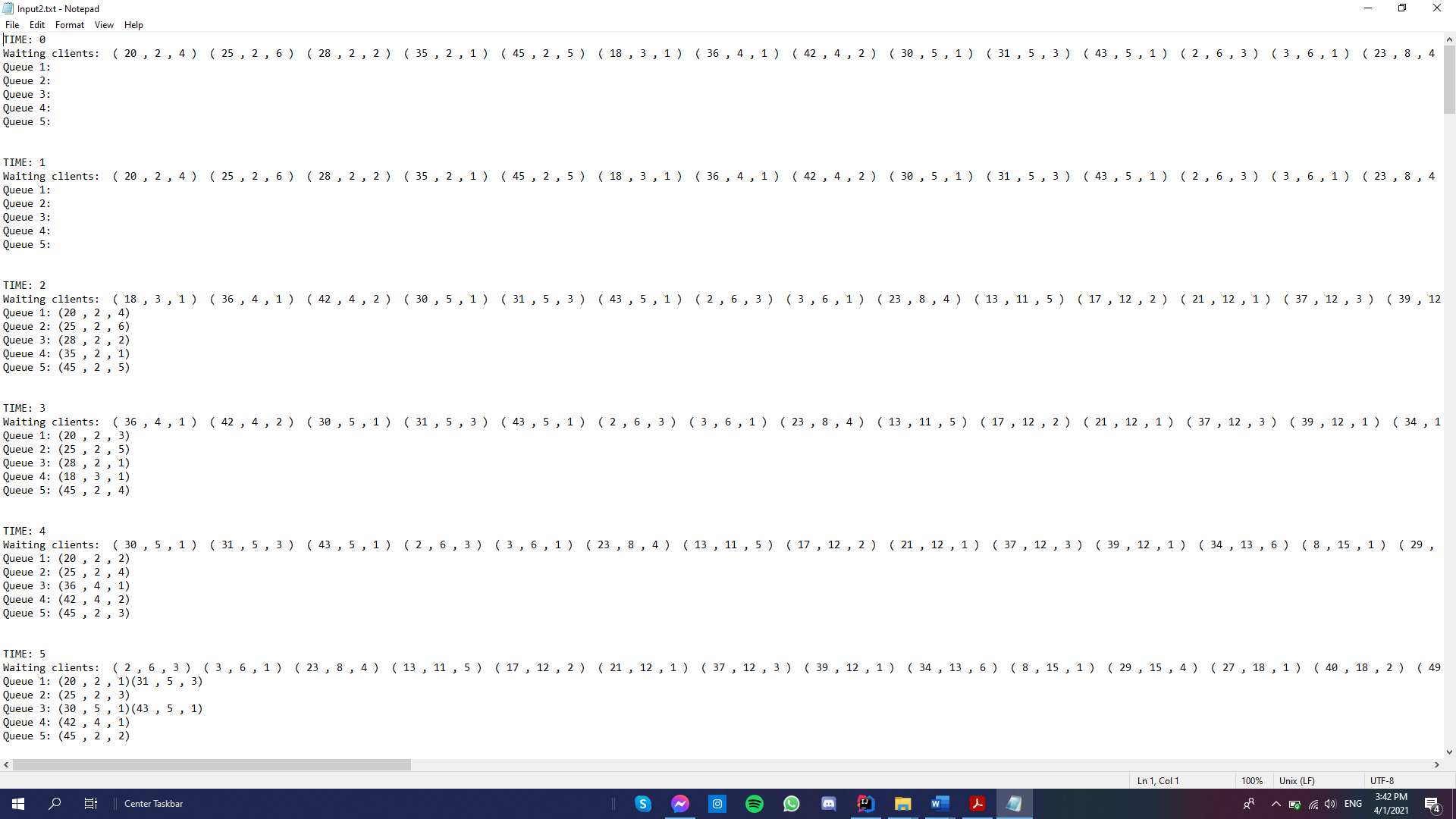
3) In metoda run se efectueaza urmatoarele obiective:

* Se deschide un FileWriter cu calea spre fisierul Results.txt
* Se intra in bucla while care verifica daca secunda curenta este mai mica decat timeLimit
* Setez in GUI timpul curent
* Parcurg lista de clienti si verific daca arrivalTime-ul a fiecarui client este egal cu timpul curent pentru a putea sa il asignez la o coada. Dupa ce apelez metoda dispatchClient ii si dau remove din coada in care se afla. Decrementez contorul buclei deoarece dupa ce dau remove toate elementele se muta cu o pozitie la stanga si s-ar putea sa imi vina un alt client care indeplineste si el aceeasi conditie
* Scriu in fisier currentTime-ul
* Imi iau un string x pentru a putea sa imi construiesc clientii ramasi. Daca nu mai am clienti care trebuie sa astepte atunci nu afisez nimic in GUI. Parcurg lista de clienti si imi afisez atat in fisier fiecare client cat imi si creez in x toti clientii ramas (pe care dupa ii afisez in GUI)
* Parcurg lista de cozi. Afisez in fisier numarul cozii. Imi adun la averageWaitingTime AtomicInteger-ul a fiecarei cozi. Parcurg lista de clienti a fiecarei cozi. Daca are processTime ul diferit de 0 atunci o sa il afisez. Daca e 0 atunci nu il mai afisez. De asemenea, mai am un contor firstClient care verifica daca sunt pe primul client din fiecare coada pentru a afisa serviceTime ul sau decrementat la fiecare currentTime.
* Dupa in variabila locala nrQ imi stochez numarul de cozi pe care le-a introdus utilizatorul. Daca a introdus o coada atunci numai prima coada o sa fie actualizata, daca a introdus 2 atunci 2 cozi o sa fie actualizate samd.
* Dupa toate acestea fac maximul de clienti care se afla in toate cozile pentru a afla peakHour (secunda la care se aflau cati mai multi clienti asignati la cozi)
* La final incrementez currentTime ul si pun thread-ul curent pe sleep timp de o secunda
* Dupa ce se termina bucla while afisez la final averageWaitingTime-ul, averageServiceTime-ul si peakHour-ul

4) Metoda getClientsInQueue are rolul de a returna un string ce contine toti clientii din coada q trimisa ca parametru in antetul metodei. Se verifica in primul rand daca sunt clienti asignati. Daca nu atunci se returneaza String-ul "". Daca sunt clienti atunci se parcurg toti clientii cozii si se contruiesc in string-ul x (luat ca variabila locala) si dupa parcurgere se returneaza x.

***5) Rezultate***

Exemplu de rezultate pentru al doilea input din prezentarea temei pentru care s-a cerut simularea



***6) Concluzii***

In concluzie, am implementat tot ce se cerea si cum se cerea in prezentarea pdf. Este de recomandat ca in GUI sa se introduca toate campurile necesare pentru a asigura functionalitatea simularii intrucat fiecare camp din GUI ce trebuie sa fie completat este la fel de important ca celelalte. Am implementat ca atunci ca o coada sa fie inchisa scrie "Closed" si de asemenea afiseaza si clientii ramasi.

***7) Bibliografie***

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

https://www.tutorialspoint.com/java/util/timer\_schedule\_period.htm

https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html