******

***Queues simulator***

**Moldovan Vlad-Madalin**

**CUPRINS:**

**1.Obiectivul temei**

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

**3. Proiectare (decizii de proiectare, diagrame UML, structure de date, proiectare clase, interfete, relatii , packages, algoritmi, interfata utilizator)**

**3.1 Decizii de proiectare**

**3.2 Diagrama UML**

**3.3 Proiectare Clase**

**4.Implementare**

**4.1 Clasa Task**

**4.2 Clasa Server**

**4.3 Clasa Scheduler**

**4.4 Clasa SimulationManager**

**5.Concluzii**

**6.Bibliografie**

**1.Obiectivul temei**

Obiectivul principal al temei este de a implementa un Queues simulator bazat pe repartizarea clientilor in cozi , folosind ca criteriu de repartizare timpul minim pe care un client ar trebui sa-l astepte la o coada . Pentru realizarea acestei aplicatii se vor folosi thread-uri . Obiectivul principal al acestei teme este de a ne familiariza cu thread-urile , felul in care acestea functioneaza si unde trebuie utilizate .

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

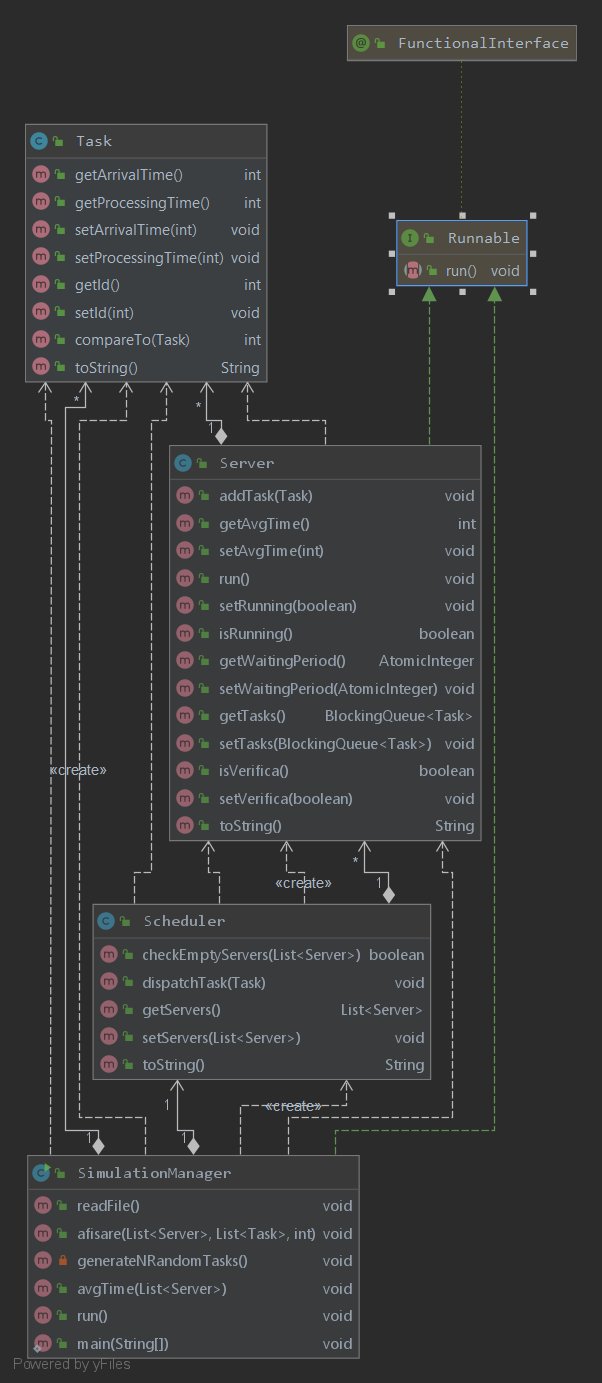
Cerinta problemei este de a implementa un Queues simulator pentru repartizarea clientilor in cozi. Se vor intoduce de numarul de clienti , numarul de cozi , timpul de simulare , intervalul de timp in care poate ajunge un client si intervalul de timp al duratei de procesare pentru un client . De exemplu :  numarul de clienti = 2 , numarul de cozi = 4 , timpul de simulare = 60 , intervalul de timp in care poate ajunge un client = 2,30 , intervalul de timp al duratei de procesare pentru un client = 2,30 . Datele vor fi introduse intr-un fisier pe linii separate , iar rezultatul va fi afisat intr-un alt fisier . Rezultatul generat va contine o lista compusa din lista de asteptare a clientilor ,time(timpul curent al executiei) si cozile (care contin sau nu clienti in functie de repartitia clientilor ) , in plus la finalul fisierului se va afisa timpul mediu de asteptare a unui client la coada .

**3.** **Proiectare (decizii de proiectare, diagrame UML, structure de date, proiectare clase, interfete, relatii , packages, algoritmi, interfata utilizator)**

**3.1 Decizii de proiectare**

A fost implementata prima data clasa “Task” care contine id-ul , arrivalTime-ul , processingTime-ul unui client . Dupa care a fost implementata clasa “Server”care continue ca attribute semnificative o lista de task-uri numita “tasks” si waitingPeriod care reprezinta timpul de asteptare din server (coada) . Dupa care a fost implementata clasa “Scheduler” care continue o lista de o lista de servere numita servers , maxNoServers , maxNoTasksPerServer , si un array de Thread-uri .La final a fost implemtata clasa “SimulationManager” de unde va fi pornit thread-ul simularii.

**3.2 Diagrama UML**

****

**3.3 Proiectare Clase**

* Clasa “Task” este utilizata pentru a stoca id-ul , arrivalTime-ul , processingTime-ul unui client . Aceasta a implementat interfata comparable pentru a facilita sortarea clientilor generate in functie de arrivalTime .
* Clasa “Server” este utilizata pentru stocarea listei de task-uri , waitingPeriod si ajuta print atributul avgTime la calcularea timpului mediu de astepare pentru fiecare client in coada pana la servire .Aceasta implementeaza interfata Runnable , datorita thread-urilor care vor fi folosit in aceasta clasa .
* Clasa “Scheduler” este utilizata in mare parte pentru stocarea unei liste de servere si pentru strategia de adaugare a unui client intr-o coada pentru ca asteptarea clientului sa fie minima .
* Clasa “SimulationManager” este utilizata pentru citirea fisierului , timparirea fisierului si pornirea thread-ului simularii . Aceasta a implementat interfata Runnable ,datorita thread-ului care va fi folosit in aceasta clasa .

**4.Implementare**

**4.1 Clasa Task**

Clasa Task este utilizata pentru a stoca datele unui client si pentru a putea forma liste de clienti . Aceasta contine urmatoarele atribute : “id” de tipul int care reprezinta id-ul clientului , “arrivalTime” de tipul int care reprezinta timpul la care a ajuns clientul si “processingTime” care reprezinta timpul pe care clientul trebuie sa il astepte pana acesta va fi servit .

Metode:

* public Task(int id,int arrivalTime, int processingTime)

-reprezinta constructorul clasei care atribuie id , arrivalTime , processingTime unui obiect nou instantiat

* public int getArrivalTime()

-reprezinta un getter prin care putem obtine ArrivalTime-ul unui obiect

* public int getProcessingTime()

-reprezinta un getter prin care putem obtine ProcessingTime-ul unui obiect

* public void setArrivalTime(int arrivalTime)

-reprezinta un setter prin care putem seta ArrivalTime-ul unui obiect cu o anumita valoare

* public void setProcessingTime(int processingTime)

-reprezinta un setter prin care putem seta ProcessinglTime-ul unui obiect cu o anumita valoare

* public int getId()

-reprezinta un getter prin care putem obtine id-ul unui obiect

* public void setId(int id)

-reprezinta un setter prin care putem seta Id-ul unui obiect cu o anumita valoare

* public int compareTo(Task c)

-aceasta metoda este scrisa cu @Override pentru a putea sorta o lista de tipul Task , metoda este pentru ordonarea crescatoare a unei liste in functie de ArrivalTime

* public String toString()

-aceasta metoda este scrisa cu @Override pentru a putea returna detalii in formatul pe care ni-l dorim

**4.2 Clasa Server**

Clasa Server reprezinta Coada la care un clien va fi repartizat . Aceasta continue urmatoarele attribute : tasks ( de tipul BlockingQueue ) care reperzinta o lista de task-uri (clienti) , waitingPeriod ( de tipul atomic integer ) care reprezinta timpul de asteptare al cozii pana aceasta va fi goala , doua variabile de tip Boolean running and verifica care ne vor ajuta la intretinerea thread-ului ( vor determina cand thread-ul este activ , doarme sau este inchis ) , avgTime ( de tip int ) care ne va ajuta sa determinam timpul mediu de asteptare al unui client pana acesta va fi servit.

Metode:

* public Server ()

- reprezinta constructorul clasei care va initializa atributele cu valorile nule

* public void addTask( Task newTask )

-reprezinta o metoda care va adauga un nou task in lista de task-uri tasks al sever-ului si va seta variabila running pe true pentru a ca conditia de rulare a thread-ului sa fie active , waitingPeriod va fi incrementat cu processing Time-ul task-ului adaugat , iar la avgTime ii va fi adunata noua valoare a lui waitingPeriod

* public int getAvgTime ()

-reprezinta un getter prin care putem obtine avgTime-ul unui obiect

* public void setAvgTime ( int avgTime )

-reprezinta un setter prin care putem seta valoarea avgTime a unui obiect

* public void run ()

-metoda run este chemata atunci cand un thread este pornit . Aceasta verifica daca lista de task-uri (tasks) nu este goala (adica daca avem clienti la coada) in cazul in care nu este goala aceasta va lua primul element al cozii , thread-ul va intra in modul sleep cate o secunda atata timp cat processing time-ul primului element din coada este diferit de 0 ( in metoda run() din SimulationManager -> processing time-ul primului element din coada si waitingPeriod vor fi decrementate la fiecare secunda pana cand el va fi scos din coada) dupa care elemental va fi scos din coada .In cazul in care lista este goala atunci thread-ul va intra in modul sleep pana cand un nou element va fi introdus in coada sau simularea sa terminat , iar thread-ul va fi inchis.

* public void setRunning(boolean running)

-reprezinta un setter prin care putem seta valoarea running a unui obiect

* public boolean isRunning()

-reprezinta un getter prin care putem obtine running-ul unui obiect

* public AtomicInteger getWaitingPeriod()

-reprezinta un getter prin care putem obtine waitingPeriod-ul unui obiect

* public void setWaitingPeriod(AtomicInteger waitingPeriod)

-reprezinta un setter prin care putem seta valoarea waitingPeriod a unui obiect

* public BlockingQueue<Task> getTasks()

-reprezinta un getter prin care putem obtine tasks-ul (lista de task-uri) unui obiect

* public void setTasks(BlockingQueue<Task> tasks)

-reprezinta un setter prin care putem seta valoarea waitingPeriod a unui obiect

* public boolean isVerifica()

-reprezinta un getter prin care putem obtine valoarea lui verifica a unui obiect

* public void setVerifica(boolean verifica)

-reprezinta un setter prin care putem seta valoarea verifica a unui obiect

* public String toString()

-aceasta metoda este scrisa cu @Override pentru a putea returna detalii in formatul pe care ni-l dorim

**4.3 Clasa Scheduler**

Clasa Scheduler contine urmatoarele attribute : servers ( de tipul List<Server> ) care contine o lista de servere , maxNoServer (de tipul int) , maxTasksPerServer (de tipul int) , t ( de tipul Thread[] ) care continue array-ul de thread-uri din aplicatie .

Metode:

* public Scheduler(int maxNoServers,int maxTasksPerServer)

-reprezinta constructorul clasei ca va initializa atributele maxNoServers si maxTasksPerServer cu valorile transmise ca parametru si se vor initializa un numar de thread-uri egal cu numarul de elemente din lista servers

* public boolean checkEmptyServers(List<Server> s)

-reprezinta o metoda care verifica daca toate serverele sunt empty (daca cozile nu au clienti) si va returna true in caz ca este adevarat , iar in caz contrar va returna false

* public void dispatchTask(Task task)

-reprezinta metoda prin care clienti sunt trimisi la coada corespunzatoare (cu cel mai mic timp de asteptare) . Se va cauta server-ul (coada) care are cel mai mic waitingPeriod , dupa care se va adauga in server-ul respective task-ul (clientul) care trebuie introdus

* public List<Server> getServers()

-reprezinta un getter prin care putem obtine lista “servers” a unui obiect

* public void setServers(List<Server> servers)

- reprezinta un setter prin care putem seta lista “servers” a unui obiect

* public String toString()

-aceasta metoda este scrisa cu @Override pentru a putea returna detalii in formatul pe care ni-l dorim

**4.4 Clasa SimulationManager**

Clasa SimulationManager continue urmatoarele attribute : timeLimit ( de tipul int ) , maxProcessingTime (de tipul int) , minProcessingTime (de tipul int) , minArrivalTime (de tipul int) , maxArrivalTime (de tipul int) , numberOfClients (de tipul int) , numberOfServers(de tipul int) , input (de tipul String) reprezinta denumirea fisierului de input de unde se vor citi datele de intrare , output (de tipul String) reprezinta denumirea fisierului de output unde se vor tipari rezultate , Scheduler ( de tipul Scheduler) , generatedTasks (de tipul List<Task>)

Metode:

* public void readFile() throws FileNotFoundException

-reprezinta o metoda prin care se vor citi din fisierul de input atributele urmatoare: timeLimite , maxProcessingTime , minProcessingTime , minArrivalTime , maxArrivalTime , numberOfClients , numberOfServers

* public SimulationManager(String input,String output) throws FileNotFoundException

-reprezinta constructorul clasei unde se vor apela metodele pentru citire si generare a clientilor

* public void afisare (List<Server> a,List<Task> list,int time) throws IOException

-reprezinta o metoda pentru afisarea in fisier a rezultatelor sub forma din cerinta

* private void generateNRandomTasks()

-reprezinta o metoda pentru generarea random a clientilor care respecta constrangerile care au fost introduse in fisierul de input

* public void avgTime(List<Server> s) throws IOException

-reprezinta o metoda pentru afisarea timpului mediu de asteptare a unui client in coada pana cand acesta va fi servit , afisarea se va face la finalul fisierului exact ca in cerinta

* public void run()

-metoda run() este chemata atunci cand thread-ul este pornit . Conditia de rulare a thread-ului este currentTime < timeLimit , unde currenTime va reprezenta timpul current care incepe de la 0 si va fi incrementat in while . Se va cauta in lista de clienti generata random daca exista un client care are timeArrival-ul egal cu currentTimp in cazul in care este adevarat se va apela metoda dispatchTask pentru a repartiza clientul intr-o coada . Dupa care se apela functia de afisare si se vor decrementa waitingPeriod si processigTime-ul primului element din coada , curentTime va fi incrementat se va verifica daca continuarea simularii mai are rost adica daca currentTime este egal cu TimeLimit (atunci se vor opri toate thread-urile din aplicatie) sau daca cozile sunt goale si nu mai mai exista client in lista de asteptare ( la fel se vor inchide toate thread-urile ) . La fiecare pas in while la final thread-ul va fi pus in modul sleep pentru o secunda .

* public static void main(String[] args) throws IOException

-reprezinta metoda main de unde se va porni thread-ul simularii si tot de aici se vor lua prin intermediul args[0] , args[1] fisierele de input respective output

**5.Concluzii**

A fost implementat un Queues Simulator cu ajutorul thread-urilor pentru a realiza o sincronizare cat mai buna a aplicatiei . Din aceasta tema am invata sa lucrez cu un thread si utilitatea acestora intr-o aplicatie . De asemenea am invatat sa creez un jar file pentru rularea aplicatiei .

**6.Bibliografie**

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

2. <https://www.tutorialspoint.com/java/util/timer_schedule_period.htm>

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

4. Cursul de Thread-uri din semestrul trecut de la Programare Orientate pe Obiect - <http://users.utcluj.ro/~igiosan/Resources/POO/Curs/POO11.pdf>

5. Structura aplicatiei a fost inspirata din pdf-ul care ne-a expicat cum ar trebui sa arate structura aplicatiei