DOCUMENTATIE

Tema numarul 2

Popovici Eusebiu-Ionut

Grupa 30224

CUPRINS

1. Obiectivul temei

Obiectvul principal al temei este acela de a realiza implementarea unei aplicatii ce reuseste in mod eficient sa aranjeze clientii la cozi(ca la magazine),rezultatele putand fi vazute in timp real in interfata grafica dar si in interiorul unui fisier.

Obiectivele secundare sunt reprezentate de:

-Analiza problemei,modelare,scenarii,cazuri de utilizare.

-Proiectare: vor fi prezntate modul in care a fost proiectata aplicatia in OOP(MVC),diagrama UML de clase si pachete,structurile de date folosite,interfetele definite si algoritmii folositi.

-Implementare: descrierea fiecare clase cu campuri si metode.

-Rezultate: prezentarea rezultatelor afisate in interfata dar si in fisierul txt.

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

Analiza problemei

-Aplicatia lasa utilizatorii sa introduca datele pentru simulare

-Aplicatia foloseste o strategie de timp pentru a introduce cat mai eficient clientii in cozi

-Aplicatia lasa clientii sa porneasca simularea prin intermediul unui buton

-Aplicatia afiseaza in timp real timpul,clientii care urmeaza sa intre in coada si continutul cozilor,atat cu clienti cat si goale

-Aplicatia afiseaza la finalul simularii rezultatele obtiunte

Modelare

Am folosit conceptul de thread pentru a putea observa aceste rezultate in timp real in interfata noastra grafica.Am pornit un thread principal,iar dupa am folosit conceptul de multi-threading,unde am mai pornit cate un thread pentru fiecare coada

Scenarii

Fiecare scenariu de utilizare respecta,aproximativ,acelasi caz de utilizare,diferenta reprezentand-o datele de intrare,ce pot sa difere de la o rulare la alt

Utilizatorul trebuie sa introduca datele pentru numarul de clienti,numarul de cozi,intervalul in care clientii generati aleatoriu se vor pune in coada,intervalul in care clientii generati aletoriu vor fi serviti si timpul total de simulare al aplicatiei,in secunde.

In caz ca vor fi introduse date eronate de la tastatura,va aparea un mesaj de eroare.

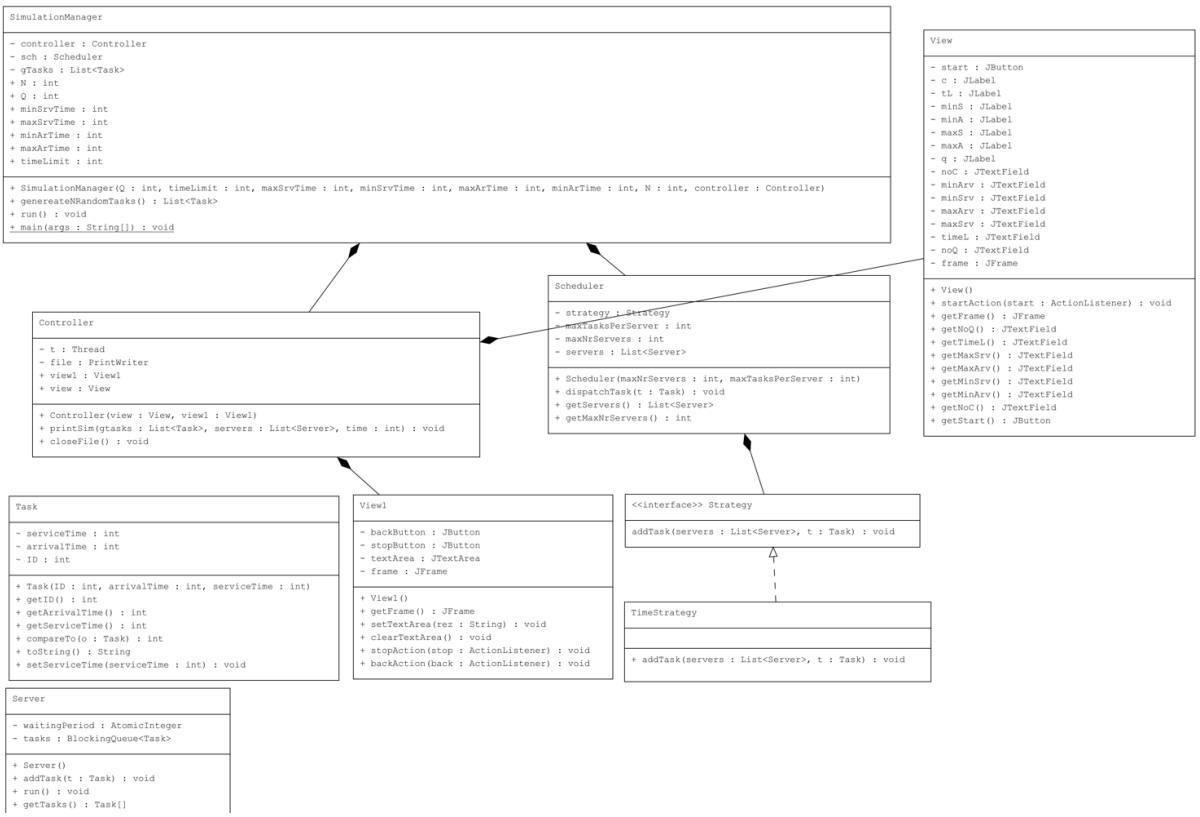
Cazuri de utilizare

La apasarea butonului de “START”,dupa introducerea datelor,se va deschide o noua interfata grafica in care vom vedea rezultatele dorite,iar tot in acelasi timp,prin intermediul OOP-ului vom executa si partea de back-end necesara.

3.Proiectare

Proiectarea OOP a aplicatie va aborda principiile programarii orientate pe obiecte,totul fiind incapsulat in clase,care se vor afla in diferite pachete pentru ca proiectul sa fie usor de citit si de inteles(Pachetele: GUI, Logical, Model).

Diagrama UML va fi folosita pentru a arata relatiile dintre clase si pachete.



Structurile de date folosite

Fiecare client este generat random pentru simularea aplicatiei,fiind pusi intr-un List de clienti.Pentru cozile generate s-au folosit BlockingQueue in care au fost pusi clientii,acest tip de structura asigurand thread safe proiectului.Tot pentru a fi thread safe am folosit si structura de date AtomicInteger pentru a transmite timpul pe care trebuie sa-l astepte un client care intra in coada.Pentru a stoca cozile pe care le vom folosi in proiect,am folosit tot un List de cozi.

Algoritmii utilizati

In cadrul modelului am implementat algoritmul pentru a folosi o coada ca un thread. Fiecare client asteapta la coada pana cand ii vine randul. In tot acest interval, timpul de asteptare este updatat de catre coada. Dupa ce un client a fost procesat, acesta este eliminat din coada sa.

In cadrul controller-ului am folosit algoritmi asemanatori pentru a pune un client in coada in functie de strategia aleasa. Pentru strategia in care este pus la coada cu clienti cei mai putini, algoritmul cauta coada cu cei mai putin client. In cazul strategie in care timpul de asteptare este minim, algorimul cauta coada in care timpul de asteptat este cel mai mic. In ambele strategii, atunci cand aceasta coada este gasita, clientul este adugat. In cadrul simularii, pentru a ne da seama daca mai sunt clienti la cozi dupa ce s-a terminat timpul de simulare, am un utilizat un algoritm ce verifica acest lucru. Pentru a aduga un client la una dintre cozi se verifica daca timpul sa de sosire este mai mic sau egal cu timpul current (mai mic sau egal deoarece poate exitsa cazul in care pot sa vina mai multi clienti in acelasi timp). De asemenea, am folosit un algoritm ce verfica daca toate cozile sunt pline. Astfel, daca ne aflam in acest caz, clientul asteapata pana cand cel putin una dintre cozi este eliberata (daca nu as fi tratat acest caz, in cadru simularii “as fi pierdut” o parte din clientini deoarece i-as fi extras din coada, dar acesti nu ar fi putut fi pusi la nicinua).

4.Implementare

In fiecare pachet se afla cel putin o clasa.

Clasa Task: Clasa Client are drept scop crearea unui nou client ce va fi ulterior pus la una dintre cozi. Acesta are trei attribute: id → reprezinta id-ul generat pentru clientul respective, serviceTime → reprezinta timpul de servire atunci cand o sa-i vina randul si curServiceTime → este folosit pentru a face update la cat timp mai are de asteptat la coada atunci cand este servit. Metoda toString() este suprascrisa pentru a afisa in mod corespunzator datele despre un client : “(id, arrival time, service time)”.

Clasa Server: Clasa Server are drept elemente o coada de clienti si un AtomicInteger reprezentand waitingPeriod-ul.Aici avem metodele addTask,unde adaugam un client in

coada,implemnteaza metoda compareTo care ne sorteaza server-ul dupa waitingPeriod-ul cel mai mic,gettere si setter dar si metoda run suprascrisa unde la un interval de timp scoatem clientii din coada si dam sleep serviceTime secunde.

Interfata Strategy si clasa TimeStrategy care o implementeaza: Aceasta interfata si aceasta clasa merg mana in mana,si adauga clientii in coada dupa cel mai scurt serviceTime,ca sa fie eficient.

Scheduler: Aceasta clasa este utilizata pentru a crea thread-urile pentru cozi si a le pune intr-o lista. Acest lucru este realizat in constructor. Metoda dispatchTask() are drept scop sa adauge un client la o coada, folosind strategia instantiata. Metoda getServers() care ne returneaza lista de cozi.

SimulationManager: Aceasta clasa este utilizata pentru a creea cadrul de simulare pentru aplicatie.In main se deschide interfata grafica,in constructor luam valorile care vor fi introduse de utilizator de la tastatura,generam random clientii si cream un scheduler cu un numar de cozi si de clienti pe care ii primim.Metoda generateRandomTask() are drept scop genrarea random a clientilor in functie de atributele numberOfClients, minServiceTime, maxServiceTime, minArrivalTime si maxArrivalTime. Aceasta metoda este utilizata in constructor.Metoda run() sunt folosite metode precum dispatchTask(),printSim() care ne afiseaza in timp real ceea ce vrem sa vedem,si dam sleep la cate o secunda.

View si View1: Aceste sunt 2 clase in care ne sunt create cele 2 interfete grafice,in care introducem datele si pornim programul si view1 in care vedem rezultatele,putem opri thread-ul principal si ne putem intoarce la prima interfata.

Controller: Aceasta clasa este practic clasa care controleaza simularea noastra,in care pornim thread-ul principal,care afiseaza atat in interfata grafica cat si in fisier,care opreste thread-ul si care inchide fisierul.Toate acestea fiind implementante in ActionListener-urile butoanelor din interfata.

5.Rezultate

Pentru a testa rezultatele am folosit mai multe exemlpe,aleatorii,pe care le-am analizat cu atentie,dar si exemple aflate in cerinta proiectului:

Test 1	Test 2	Test 3
N = 4 Q = 2 $t_{simulation}^{MAX}$ = 60 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}]$ = [2, 30] $[t_{service}^{MIN}, t_{service}^{MAX}]$= [2, 4]	N = 50 Q = 5 $t_{simulation}^{MAX}$ = 60 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}]$ = [2, 40] $[t_{service}^{MIN}, t_{service}^{MAX}]$= [1, 7]	N = 1000 Q = 20 $t_{simulation}^{MAX}$ = 200 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}]$ = [10, 100] $[t_{service}^{MIN}, t_{service}^{MAX}]$= [3, 9]

6.Concluzii

In concluzie,acesta a fost unul dintre cele mai complexe proiecte la care am lucrat,deoarece este destul de dificil sa te acomodezi cu notiunile de thread si multithreading si cum se combina acestea unele cu altele.Totodata,aceasta tema a reprezentat un prilej bun de a invata lucruri noi si a le pune in practica intr-un mod inedit.

Ca posibile modalitati de dezvoltare ulterioara,am putea conecta programul nostru java la o baze de date unde sa stocam clientii si cozile noastre,si am putea modifica cozile noastre in magazine de exemplu,cu intervale orare specifice,si angajati care lucreaza la aceste magazine si primesc si servec clientii respectivi.

7.Bibliografie

1. https://www.javatpoint.com/multithreading-in-java

2. https://www.digitalocean.com/community/tutorials/thread-safety-in-java

3. https://www.geeksforgeeks.org/runnable-interface-in-java/