MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE DEPARTAMENTUL CALCULATOARE

Sistem de procesare a cozilor

Documentație

Berea Roxana

Grupa 30226 | An 2 semestrul 2

Cuprins

1. Obiectiv

2. Analiza problemei

3.Exemplu de lucru

4.Implementare

- 4.1.Clasa Client
- 4.2.Clasa Server
- 4.4.Clasa Scheduler
- 4.5.Clasa SimulationManager
- 4.6.Interfata Strategy si clasa ConcreteTimeStrategy
- 4.7.Clasa GUI

5.Proiectare

6.Testare si rezultate

7.Concluzii

8.Bibliografie

1.Obiectiv

Obiectivul acestei teme de laborator a fost sa proiectam si sa implementam un sistem de procesare a unor cozi. Sistemul primeste un numar de clienti la anumite momente de timp, care vor fi distibuiti la cozi. Pentru ca aplicatia sa fie considerata functionala si eficiente, clientii vor fi distribuiti in asa fel incat timpul de asteptare sa fie cat mai mic cu putinta.

Sistemul va avea o interfata grafica, care poate fi utilizata usor si confortabil de catre orice utilizator, unde vor putea fi introduce: numarul cozilor, numarul clientilor, timpul de simulare, timpii de a ajunge a clientilor si timpii de procesare a unui client.

Printre cerintele obligatorii ale acestei teme se gaseste si folosirea unui thread pentru fiecare coada, implementarea unei intefete grafice, generarea aleatorie a clientilor.

Conform site-ului Wikipedia, conceptul de thread (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către sistemul de operare. Este folosit în programare pentru a eficientiza execuția programelor, executând porțiuni distincte de cod în paralel în interiorul aceluiași proces. Câteodata însă, aceste portiuni de cod care constituie corpul threadurilor, nu sunt complet independente și în anumite momente ale execuției, se poate întampla ca un thread să trebuiască să aștepte execuția unor instructiuni din alt thread, pentru a putea continua execuția propriilor instrucțiuni. Această tehnică, prin care un thread asteaptă execuția altor threaduri înainte de a continua propria execuție, se numește sincronizarea thread-urilor.

Pentru implementarea aplicatiei folosind firele de executie, este necesara implementarea interfetei Runnable sau extinderea clasei Thread. Diferenta dintre cele doua modalitati de folosire a firelor de executie este urmatoarea: Runnable este o interfata, implementata de clasele care o folosesc, acestea fiind capabile sa mostenească alte clase, pe cand Thread este o clasa parinte, care va fi mostenita, iar clasele care o mostenesc, nu sunt capabile sa extinda alte clase.

2. Analiza problemei

Analiza problemei presupune identificarea legaturilor dintre clasele proiectului, precum si functionalitatea acestuia. In acest sens, programarea orientata pe obiect ne permite implementarea aplicatiei cunoscand un numar minim necesar de informatii.

Principiul de multithreading presupune execuția mai multor thread-uri în același pipeline, fiecare având propria secțiune de timp în care este menit să lucreze. Odată cu creșterea capabilităților procesoarelor au crescut și cererile de performanță, asta ducând la solicitarea la maxim a resurselor unui procesor. Necesitatea multithreading-ului a venit de la observația că unele procesoare puteau pierde timp prețios în așteptarea unui eveniment pentru o anumită sarcină. Foarte repede a fost observat potențialul principiului de paralelizare a unui proces, atât la nivel de instrucțiune, cât și la nivel de fir de execuție. Firul de execuție sau thread-ul este un mic proces sau task, având propriile instrucțiuni și date. Ca și aplicabilitate, multithreading-ul poate fi folosit pentru sporirea eficienței atât în cadrul multiprogramării sau a sarcinilor de lucru pe mai multe fire de execuție, cât și în cadrul unui singur program. Astfel, un fir de execuție poate rula în timp ce alt fir de execuție așteaptă un anumit eveniment.

Clasele necesare pentru modelarea acestei aplicatii sunt: SimulationManager, Scheduler, Strategy, Server si Client. Additional am implementat clasele: ConcreteStrategyTime, pemtru o descriere mai detaliata a strategiei de a distribui clientii la coada la care ar avea de asteptat cel mai putin si clasa GUI, in care am realizat interfata grafica. Clasa Client ofera informatii pentru fiecare client despre timpul la care a ajuns si a fost pus la coada, timpul necesar pentru a fi servit si timpul de finalizare. In clasa Scheduler se creeaza serverele si clientii sunt impartiti la cozi dupa strategia prestabilita in clasa ConcreteStrategyTime. Clasa Server este responsabila de modelarea cozilor (thread-urilor). Interfata Strategy este implementata de clasa ConcreteStrategyTime, care pune clientii la coada la care au cl mai putin de asteptat.

3. Exemplu de lucru

Este necesara introducerea corecta a datelor de simulare, bifarea casutei pentru a alege strategia TimeStrategy, iar apoi apasarea butonului de START pentru ca aplicatia sa functioneze correct.

Astfel, numerele introduce in fiecare casuta trebuie sa fie positive sim ai mari decat 0, in caz contrar, un mesaj corespunzator va fi afisat pe ecran si datele vor trebui introduce din nou.

Programul a fost testat pe cele 3 cazuri de testare necesare pentru predarea temei a doua. Un alt exemplu pentru a testa un numar mai mic de client si cozi astfel incat sa se poata urmari in detaliu functionarea aplicatiei este urmatoul:

- N: 4
- Q: 2
- Timp de simulare: 25
- MIN arrival time: 2
- MAX arrival time: 20
- MIN serving time: 1
- MAX serving time: 5

Cazuri posibile pentru ca simularea sa esueze pot fi urmatoarele:

- Introducerea unor numere mai mici decat 0 sau litere -> apare un mesaj de eroare
- Lasarea unor campuri libere -> apare mesaj de eroare

Cazuri de success:

- Introducerea corecta a datelor ->simularea incepe si aplicatia functioneaza correct
- Introducerea datelor fara a bifa casuta pentru strategie ->simularea incepe, dar este posibil sa nu functioneze in totalitate corect

4. Implementare

Pentru inceput, am ales sa impart proiectul in pachete, respectand conventiile programarii pe obiect si grupand clasele astfel:

- Pachetul Controller:

  Continue clasa: SimulationManager

- Pachetul Model:

  Contine: clasa Client

  Clasa Server

  Clasa Scheduler

  Clasa ConcreteStrategyTime, care extinde

  Interfata Strategy

- Pachetul View:

  Continue interfata grafica: Clasa GUI

4.1.Clasa Client

Clasa Client modeleaza datele clientilor in functie de solicitarile ce vor avea loc pe cozi.

Detaliile specific si necesare pentru fiecare client sunt urmatoarele: id, timpul la care a sosit, timpul de procesare (servire) si timpul de finalizare. Cu ajutorul metodei de generare aleatorie a clientilor, id-ul acestora reprezinta un numar, ales in ordine crescatoare, iar timpul de sosire este

generat aleatory, dar in limitele introduce de la tastatura in campurile de MIN arrival time si MAX arrival time. Timpul de finalizare se calculeaza in aceasta clasa, fiind suma dintre timpul de sosire si timpul de procesare/servire.

Tot in aceasta clasa am asigurat sortarea clientilor in functie de timpul de sosire, iar pentru afisarea detaliilor clientilor am suprascris functia toString din clasa Server.

4.2.Clasa Server

Pentru a scapa de problema sincronizarii operatiilor pe coada am utilizat BlockingQueue, iar variabila instata de tipul Atomic am utilizat-o pentru controlul firelor de executie, prin pornirea, respective oprirea metodei run().

In aceasta clasa are loc simularea procesarii clientilor prin intermediu metodei run(). Metoda functioneaza in felul urmator: cat timp coada nu este goala, dup ace clientul de pe prima pozitie este scos, adica dupa ce a fost servit, thread-ul este pus pe sleep pentru un timp echivalent cu timpul de procesare al clientului inmultit cu 1000 (pentru ca utilizatorul sa poate observa procesarea clientilor in interfata grafica.

Pentru afisarea continutului fiecarei cozi, am suprascris din nou metoda toString si am ales o metoda mai scurta de a scrie reprezentativ fiecare client, alaturand initialei “c” id-ul clientului.

4.4.Clasa Scheduler

Utilizatorul va introduce in campul “Q: “ numarul de cozi droit, iar in aceasta functie vor fi initializate acle Q numar de cozi. Tot in aceasta clasa se primeste firul de executie corespunzator fiecarei cozi. Clientii vor fi adaugati in coada care indeplineste cel mai bine cerintele strategiei de timp implementate in clasa ConcreteStrategyTime.

4.5.Clasa SimulationManager

In aceasta clasa sunt interpretate datele citite din interfata grafica. Daca aceste date sunt introduce gresit vor aparea mesaje de eroare, asa cum am descris cazurile la punctul 3. Firul de executie principal este reprezentat de aceasta clasa, fiind principal pentru realizarea intregii simulari. Metoda

de generare aleatoare a clientilor se regaseste in aceasta clasa, timpii de sosire si de procesare fiind generate respactand intervalele introduce in campurile din interfata grafica.

Alte metode intalnite in aceasta clasa sunt: metoda prin care se actulaizeaza starea cozilor si o metoda prin care este calculate timpul total in care fiecare coada ramane goala.

Metoda de run incepe atunci cand este apasat butonul de start din fereastra grafica, insa simularea incepe doar daca toate datele au fost introduce correct. LA finalul acestei metode se opresc toate firele de executie si vor fi afisate int-un pop-up additional date statistice care contin informatii despre momentul de varf al simularii (Peak hour), timpul mediu de asteptare la cozi, precum si timpul in care cozile au ramas goale.

4.6.Interfata Strategy si clasa ConcreteStrategyTime

Interfata Strategy este extensa de clasa ConcreteTimeStrategy, care va cauta coada cu timpul de asteptare minim, considerand-o pe aceasta varianta optima de a repartiza noii clienti ajunsi.

4.7.Clasa GUI

Aceasta clasa, asa cum ii spune si denumirea, are rolul de a proiecta interfata cu utilizatorul. Fereastra care va aparea la rularea aplocatiei contine in partea stanga un panou cu textfield-uri corespunzatoare introducerii tuturor datelor necesare simularii, un alt panou in mijloc, in care va fi afisat statusul cozilor si clientii care se afla la coada in anumite momente de timp, acesta fiind mereu actualizat, iar in partea dreapta a ecranului se va afla cel de-al treilea panou din fereastra, “log of events”, in care sunt descrise in cuvinte actiuniile ce au loc asupra cozilor la fiecare moment de timp.

5. Implementare si testare

Pentru o functionare corecta a aplicatiei este recomandata urmarea urmatoarelor reguli de introducere a datelorr:

- Numerele introduse in campurile din partea stanga a ferestrei grafice trebuie sa fie pozitive si preferabil diferite de zero .
- Este recomandata bifarea casutei de setare a strategiei de timp.
- pentru a putea urmari un exemplu valid de testare a programului se pot introduce urmatoarele valori:

  N: 4
  Q: 2
  Timp de simulare: 25
  MIN arrival time: 2
  MAX arrival time: 20
  MIN serving time: 1
  MAX serving time: 5

- dupa ca toate campurile au fost complétate si v-ati asigurat ca datele sunt corecte, mai lipseste doar apasarea butonului “start” pentru a activa intregul sistema de procesare a cozilor.

5. Proiectare

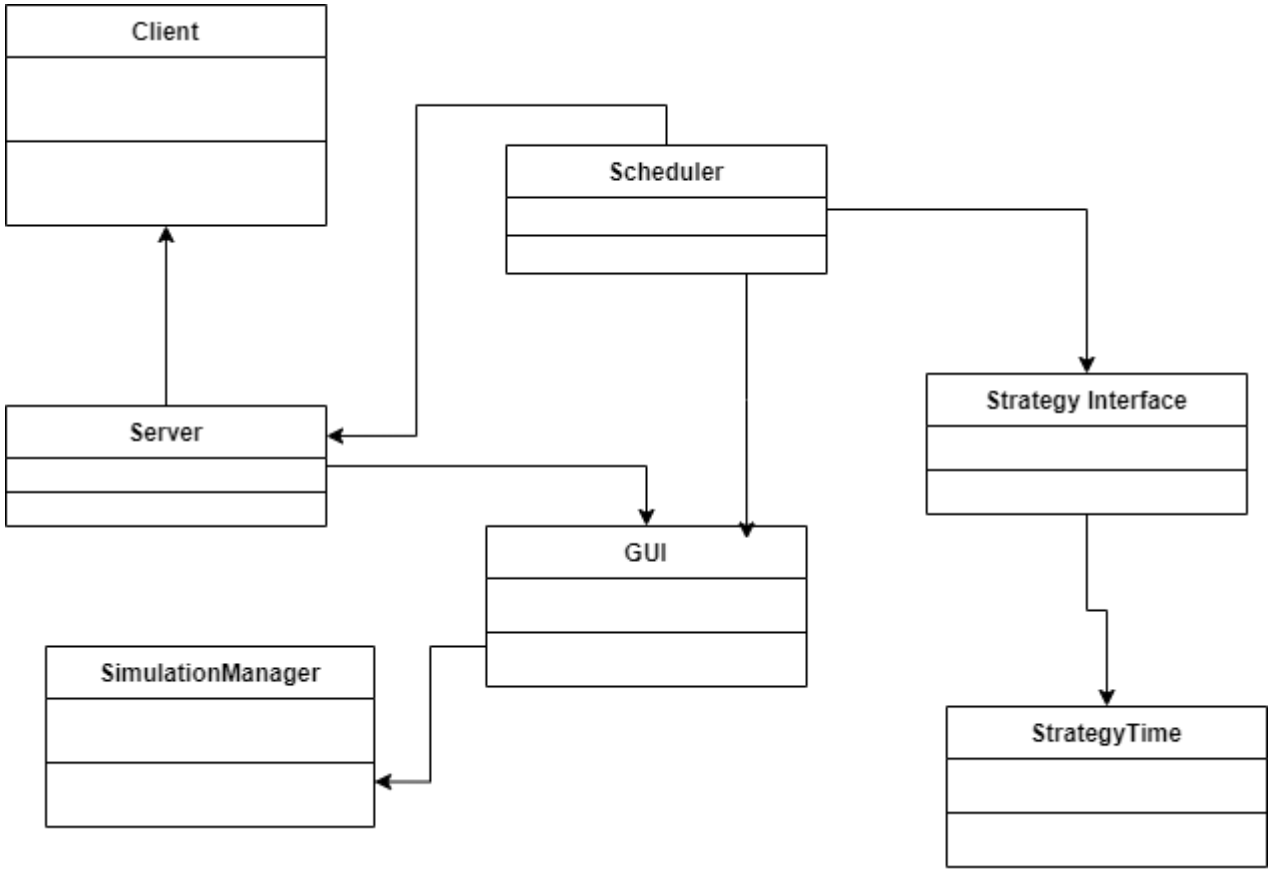
Pentru proiectarea aplicatiei am organizat programul in 3 pachete, asa cum le-am descris in Implementare la punctul 4, observandu-se paradigma Model-View-Controller.

Clasele au fost numite sugestiv si sunt implementate toate aspectele necesare bunei functionarii a simularii de procesare a clientilor si distribuiea lor la cozi.

Interfata grafica a fost conceputa folosin JFrame, reusind astfel sa implementez o GUI cu careo rice utilizator se poate familiariza in scurt timp, fiind usor de utilizat, campurile fiind numite explicit si avand buton de pornire a executiei. De asemenea este permisa navigarea is sus si in jos, respectiv stanga-dreapta pe panel-urile in care vor fi scrise detaliile despre cozi, clienti si timpii de eecutie.

Sunt atasate in proiect 3 imagini in care se pot observa calculate: momentul de timp la care a fost cel mai aglomerat, timpul mediu de asteptare, timpul mediu de servire, numarul total de clienti, acestea nefiind afisate la sfarsitul log-ului de evenimente.

Diagrama UML de clase arata in felul urmator:

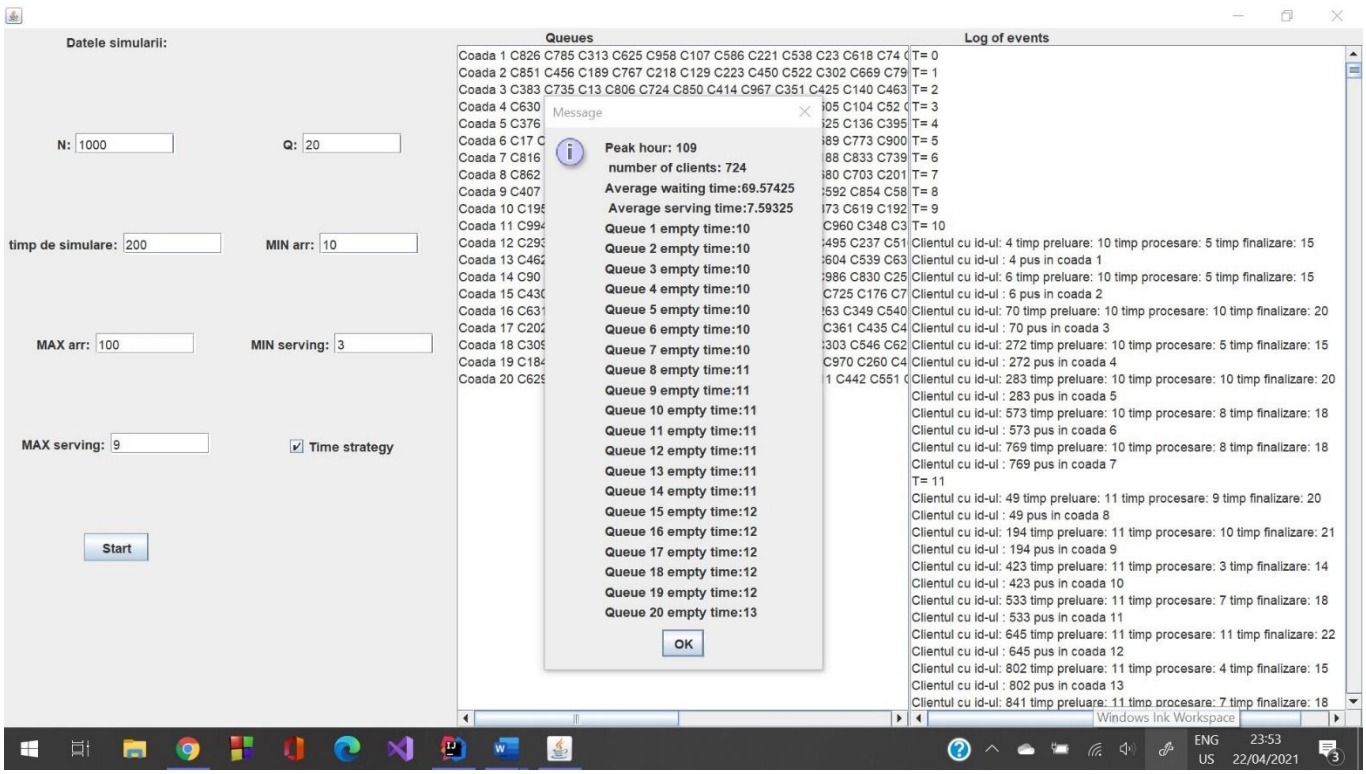


La realizarea diagramei de clase un am mai reusit sa scriu metodele si variabilele folosite in fiecare clasa inainte de a incarca assignment-ul pe GitLab.

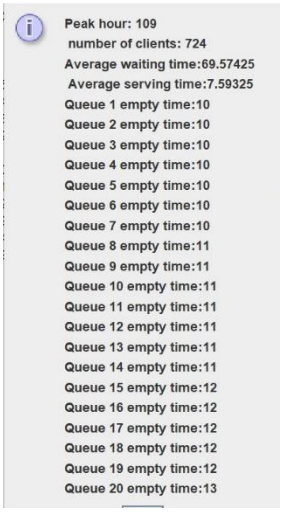
Am utilizat ArrayList si BlockingQueue drept structuri de date pentru stocarea clientilor ordonati in functie de timpul la care au ajuns, respectiv, structura BlockingQueue, pentru a elimina si pentru a adauga clienti in cozi fara sa mai fie necesara sincronizarea metodelor.

6.Testare si rezultate

Regulile pentru testarea aplicatiei au fost descrise in detaliu la punctul 3, unde am dat si un exemplu de introducere a datelor. Aici voi atasa o imagine in care se poate observa felul in care vor fi afisate rezultatele finale.



Rezultatele pentru cele trei teste cerute se pot gasi in fisierele text incarcate. Datele care s-au cerut pentru a fi afisate la finalul fisierelor se pot observa doar in cele 3 poze, in fereastra “Message” din mijlocul ecranului de aceea voi atasa in documentatie imaginile cu rezultatele celor 3 teste.



7.Concluzii

In concluzie, prin realizarea acestui proiect mi-am fixat mult mai bine informatia invatata in primul semestru la programarea orientata pe obiect, dar am vazut si metode noi de realizare a interfetei grafice si de testare a programului si de asemenea, acest assignment m-a ajutat s ama familiarizez cu lucrul pe fire de executie.

Pot fi aduse imbunatatiri la aceasta aplicatie, atat in implementarea operatiilor, cat si la aspectul interfetei grafice si la posibilitatea introducerii polinoamelor intr-un mod mai la indemana utilizatorului. Dar am reusit intr-un final sa duc proiectul pana la capat, documentandu-ma chiar si de lucruri elementare ale programarii orientate pe obiect, cat si din proiecte realizate si distribuite de alte persoane cu experienta in domeniu.

8.Bibliografie

- Resursele de curs si laborator
- Oracle’s java documentationhttps://docs.oracle.com/javase/tutorial/java/index.html
- Wikipedia- despre fire de excutie
- Youtube tutorials
- Github projects
- Stackoverflow