Assignment 2
Lucru cu THREADURI

1. Obiectivul temei

Obiectivul acestei teme consta in prezentarea unei aplicatii care sa minimizeze timpul de asteptare la cozi a unor client. Ca idee generala, procesul principal care manipuleaza procesarea clientilor este reprezentat de Server. Munca lui se bazeaza pe procesarea clientilor astfel ca poate avea mai multe fire de executie. Asta presupune ca mai multi clienti pot fi procesati simultan si independent. Practic, aceasta aplicatie reflecta ideea de baza a unui supermarket, cand clientii isi fac cumparaturile, urmand apoi sa aleaga una dintre casele de marcat pentru a li se procesa cosul de cumparaturi. Fiecare angajat ce proceseaza clienti (sta la casa de matcat si analizaeaza fiecare produs din cosul clientului) este considerat ca avand un Task, o cerinta. Aceea de a « procesa » clienti .

Avand in fata exemplul dat mai sus, putem sustrage niste idei principale ce definesc obiectivul aceste teme si anume : modalitetea in care un sistem proceseaza informatia , eficienta si dexteritatea lui.

Aplicatia ar trebui sa simuleze repartizarea seriilor de clienti, care vin la un timp aleator, la cozile cele mai putin ocupate, cat si procesarea fiecarui client in parte. Mai tarziu vom vorbi si despre generarea aleatoare a clientilor avand in vedere ca momentan e un subiect mai putin important.

Vom vorbi in continuare despre modul in care am analizat eu problema.

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

In urmatoarele paragrafe (avand in vedere ca trebuie sa atingem limita de cuvinte) vom vorbi pe larg despre modul in care am “gandit” aplicatia prezentata in aceasta documentatie. Asadar vom lua fiecare component important ain parte si vom dezbate problemele intalnite pe parcursul proiectarii si programarii acesteia.

Prima data as dori sa va vorbesc putin despre modul in care m-am gandit sa imi generez clientii. Pe la laborator, pe la cursi si, de asemenra, in discutiile purtate cu anumiti colegi am incercat sa-mi exprim ideea mea. Se vorbea destul de mult despre generarea tuturor clientilor la inceput, insa eu am fost dezamagit de acesta idee. Ganditiva la urmatoarea idee : « Proprietarul magazinului stie inca de dimineata ce clienti o sa vina si cat o sa dureze procesarea fiecarui client in parte ? Nu. » Nu mi-a placut ideea asta deoarece, daca tu iti sti toti clientii inca de la inceput, atunci poti sa-i sortezi si sa-i repartizezi la cozi la secunda, incainte ca aplicatia sa inceapa treaba. Astfel tu vei sti locul fiecarui client in coada. Poti genera statisticile inca de la inceput. Alt dezavantaj…sa u nu neaparat dezavantaj, cat o distantarea de realitate. Asadar, ideea mea de a genera clienti a fost prin generarea fiecarui client in parte, pe

parcursul evolutiei cozilor. Fiecare client va fi generat la un moment dat de timp, cu un anumit timp de procesare, pe urma se va genera alt client si tot asa . Totul se va petrece pe parcursul evolutie cozilor.

Pentru generarea clientilor am nevoie de doua variabile, doi timpi pe care ii voi genera aleator : timplul pana cand apare urmatorul client, si timplul de procesare a clientului curent.

Urmatorul aspect despre care doresc sa va vorbesc este de fapt o explicatie prin care voi deslusi treaba serverului. El practic va primi clienti la momente diferite de timp . Treaba lui este sa ii repartizeze pe acestia la cea mai libera coada. Aici intervine problema procesarii clientului. Noi stabilim la inceput cate Resurse(case de marcat sau « task »-uri, numite de mine) avem care pot procesa clienti. Treaba fiecarui Task este doar de a procesa clienti. El va contine o lista de Clienti care sunt adaugati de catre server si treaba lui este doar de a-i procesa. Dupa ce un client este « procesat », el este sters din coada.

Serverul practiic gestioneaza chestia asta : adaugarea si stergerea unui client dintr-o anumita coada . El stie tot ce se intampla in spate. El contine cozile si stie de treaba fiecarei cozi. Fiecare client repartizat la o coada trece mai intai prin mana serverului. Acum ca am ajuns la sapte sute de cuvinte (sper sa nu se supere doamna profesoara daca citeste asta) o sa dezbatem o alta ideee destul de interesanta si anume modalitatea prin care facem legatura dintre acest model (server, task-uri si generarea aleatoara a clientilor) si Vederea . M-am gandit destul de mult la faptul ca nu ar trebui sa stie foarte mult unul de celalalt(modelul de vedere) si astfel am ajuns la concluzia ca vederea trebuie sa primeasca doar niste « mesaje » ce trebuie afisate pentru ca utilizatorul sa intelega ce se intampla. Nu e nevoie sa stie ce se intampla la nivel de cod, intern, dar utilizatorul ar trebui sa vada cozile : adaugarea si stergerea unui client din coada. Tot ce este legat de server este doar la nivel abstract. Practic pentru client doar cozile sunt vizibile. Asa am stabilit ca vederea sa contina cozile si gestionearea lor in timp real. Pe langa cozi am adaugat niste mesaje, pe care le voi retine si intr-un logger, care va putea fi ulterior analizat.

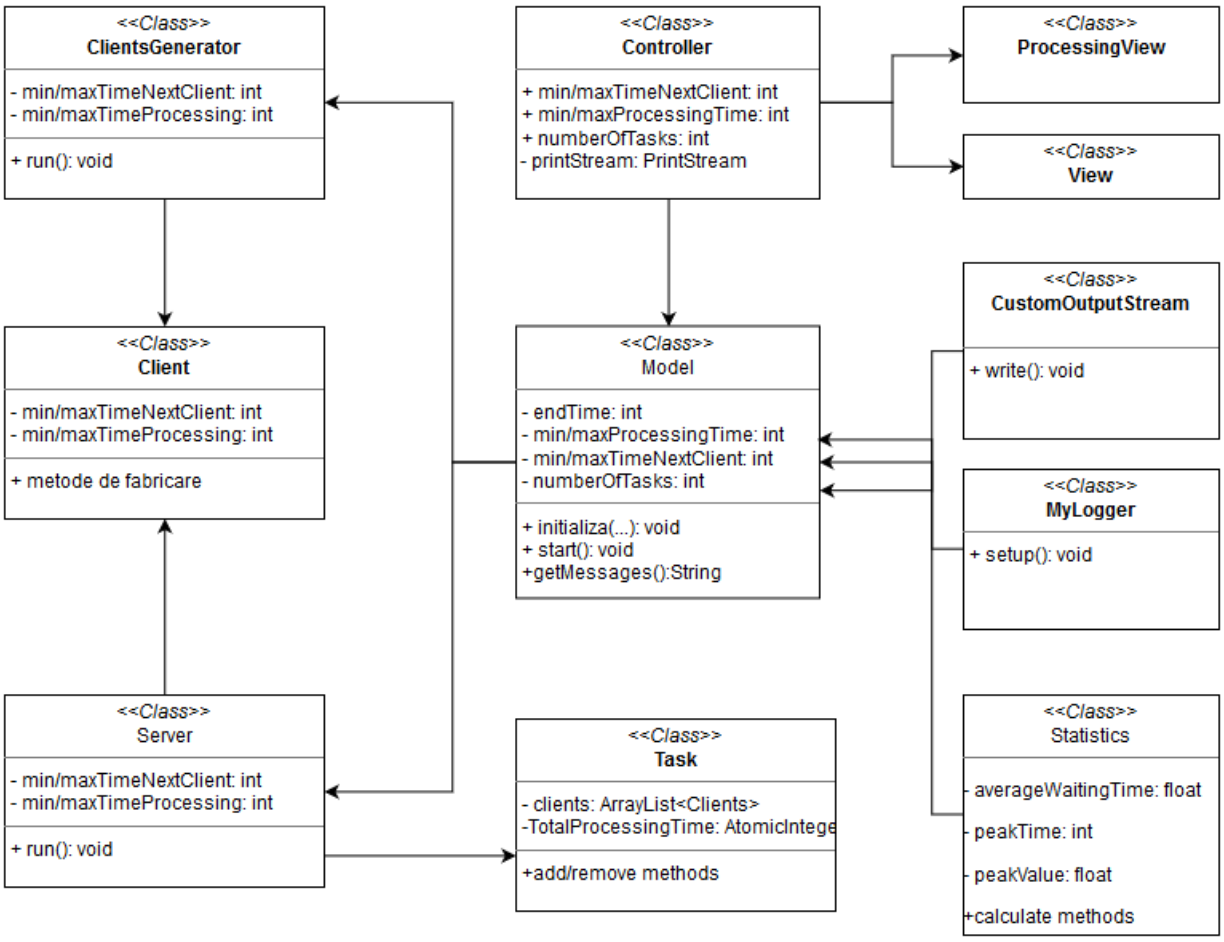
Mai am de asemenea concepute niste statistici, dar la nivel fooarte minimal. Este vorba despre timpul mediu de asteptare a unui client si de retinerea orei de varf, in care toate casele de marcat (impreuna) ating varful in ceea ce priveste timpul de procesare a clientilor. Deci practic este un timp mediu pe client, si timpul mediu de procesare pe casa, in momentul orei de varf.

Aici am avut niste dubii, pentru ca, pe anumite teste, timplu mediu de procesare a unui client era mai mic decat timpul midiu de procesare a unei cozi, in momentul de varf. Concluzia era ca se poate ca timplu mediu de procesare a unui client sa fie mai mare (acesta fiind calculat ca timpii de procesarea a tuturor clientulor impartit la numarul de clienti) decat timpul mediu de procesarea a unei cozi in momentul orei de varf datorita faprului ca in ora de varf, s-ar putea ca o casa sa aiba mai mult de lucru decat celelalte prin faptul ca acesti clienti generati random nu sosesc deodata. Astfel degeaba timpul de procesarea mediu a unui client este de 10 secunde daca un client apare in coada la 5 secunde unul fata de altul, pentru ca deja trece jumatate din timpul de procesarea a clientului anterior. Atat despre asta, sa nu ne lungim pentru ca e destul de greu de explicat chestia asta. Daca aveti intrebari cred ca fata in fata o sa va argumentez mult mai bine aceasta idee.

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

Legat de deciziile de proiectare tin sa mentionez ca am vorbit despre majoritatea acestora mai sus, si astfel o sa va explic amanuntit doar deciziile care chiar merita atentie.

Acum am sa va prezint diagram UML (foarte generalizata):



Avand in vedere faptul ca nu am marca Clasa Thread o sa mentionez aici care sunt firele de executie pe care eu le-am considerat ca fiind potrivite pentru a satisfice conditiile apricatiei.

Serverul este un Thread pentru ca el este ca un fel de manager, el controleaza tot ce se in tampla pe martea de backend. Serverul contine de asemenea o lista de task-uri . Fiecare Task este un theread separat, fiecarea lucreaza independent unul de altul. Treaba lor este doar sa proceseze clientii.

Clasa ClientsGenerator este de asemenea un Thread separat care doar imi genereaza clientii si ii da mai departe serverului care se va ocupa de ei : ii va repartiza la cea mai libera coada, urmand a fi procesati. Clasa Statistics imi genereaza la final statisticile, pe baza intamplarilor de pe durata procesului (

aplicatiei). Clasa MyLogger practic imi asigura faptul ca, dupa terminarea aplicatiei o sa am evenimentele principale stocate intr-un fisier separat. Aceasta se intampla pentru cazul in care dorim sa analizam datele mai departe, sa dezbatem anumite strategii.

Clasa CustomOutputStreamer imi redirectioneaza anumite mesaje inspre vedere astfel ca, pe langa modul vizual de a percepe actiunile petrecute pe parcursul procesului principal, vom avea afisate si niste propozitii ce descriu aceste intamplari / actiuni.

4. Implementare

In ceea ce priveste implementarea, am adoptat patternul MVC. Astfel modelul se ocupa de toata partea care munceste, adica generarea clientilor, modul de repartizarea a acestora in cozi si procesarea lor.

Vederea doar primeste mesaje de la model, si le afiseaza, nestiind cu adevarat ce se petrece. Controllerul face legatura dintre model si vedere. El este distribuitorul de mesaje.

Tin sa mentionez ca am incercat sa respect paradigmele orientarii pe obiecte si principiile SOLID. Au fost unele cazuri in care nu am avut de ales, de exemplu nu vedeam rostul mostenirii in proiectul meu, dar poate pe un proiect mai bine dezvoltat ar prinde bine aceasta.

Cel mai greu lucru a fost trecerea aceasta de la model la vedere, poate datorita faptului ca inca nu m-am obisnuit destul cu acest pattern MVC. Ma gandeam tot timpul sa izolez cat de bine pot modelul de vedere si de fiecare data ma izbeam de o alta problema. Pana la urma am ales redirectarea anumitor mesaje de la model, din consola intr-un JTextArea. Deoarece dine a nu puteam sa preiau multe date relevante, dar a fost foarte utile pentru afisarea informatiilor despre situatia cozilor si actiunile efectuate pe ele.

Ca sa fie clar, am delimitat interfata grafica in 2 JFrame-uri. Una este pentru inceput, pentru colectarea datelor, a parametrilor de functionare a aplicatiei precum numarul de cozi timpii minim si maxim de procesare a clientilor si pentru aparitia clientului urmoator (acestia fiind generati random), timpul final (practic el ne spune cat timp aplicatia noastra sa ruleze, sa proceseze clienti, sa generezze clienti).

Un caz pe care nu am avut timp sa-l tratez a fost momentul in care aplicatia se termina datorita timpului final dat ca intrare, iar in cozile mai au de procesat clienti. As fi dorit ca timpul final sa fie relativ, adica daca mai sunt clienti de procesat, sa nu se opreasca aplicatia pana cand cosile ifi vor fi terminat toata treaba. Dar treuia sa tin cont si de generarea clientilor. Acest Thread trebuia totusi sa se termine in timpul final, dar ca intrare. Sau inainte de generearea clientilor puteam verifica cat timp magazinul/serverul va mai fi deschis, astfel incat urmatorul client sa nu fie generat daca timpul lui de procesare depaseste timpul final.

Avand in vedere ca tema curenta urmareste mai mult munca cu firele de executie, consider ca am invatat modalitatea lor de functionare. Inca nu am deslusit pe deplin modalitatea lor optima in care pot fi utilizate. Recomandat cred ca ar fi sa te gandesti la idea ca trebuie sa-ti pui procesorul 100% la munca.

Adica daca un thread ocupa 20% din capacitatea curenta a procesorului, sigur mai poti evidentia un thread care sa execute alte operatii utile.

5. Rezultate

Sper sa fie nota 10. Glumesc, chiar nu sunt relevante notele pentru mine, cat este bucuria de a descoperi si invata lucruri noi. Nu stiu ce sa scriu despre rezultate. Oare va asteptati sa scriu ce statistici am obtinut in ceea ce priveste timpii medii de procesare a clientilor sau orele (secundele) de varf ? Nu prea cred.

Rezultatele sunt ca am invatat lucruri foarte interesante despre modul in care functioneaza threadurile. Tin aici sa va mentionez o intamplare de pe parcursul invatarii mele. Am gasit un tutorial pe youtube in care se prezentau doua threaduri ce incrementau acelasi counter. Practic in metoda rune ra un for de la 1 la 10.000 in fiecare thread. Deoarece am auzit de sincronizarea threadurilor inainte, si stiam ca orice thread al unui proces lucreza pe aceeasi zona de memorie a procesului, nu ma asteptam ca acel counter sa ajunga la valoarea de 20.000, datorita nesincronizarii threadurilor si a memoriei. Un fapt foarte ciudat ce mi-a dat de gandit a fost ca pe unul din teste nici macar nu a ajuns la 10.000 si atunci am ramas cu un semn de intrebare. Foarte ciudata situatia.

Acum sa discut despre cauza acesteia. Practic consideram incrementarea ca facandu-se in trei etape : citire din memorie C, incrementare I si scriere inapoi in memorie S.

In urmaotarele 2 liniiva prezin separate o potenntiala evolutie a threadurilor noastre (consideram incrementarea pana la 3 doar). Avem astfel:

Thread1: …C…I…S…C.I.S……C……I.....S

Thread2:..C……I……S……C……I……..S..C…I…S..

Presupunand ca la inceput este citita din memorie valoarea 1, ea va trebui sa ajunga cel putin la 3 pentur ca este incrementata de 3 ori de fiecare thread. Problema apara cand Thread-ul 1 citeste valoarea si o incrementeaza, apoi iar o citeste , o incrementeza si o scrie inapoi in memorie (scrie valoarea 3) iar threadul 2 deoarece a fost putin mai lent, va scrie in memorie valoaorea 2 dupa ce primul thread a scris 3. Astfel ca Primul Thread va incerca sa citeasca din noud in memorie , dar va gasi valoarea 2 si nu 3, pe care o va incrementa si o va scrie ultima data ca fiind 3. Astfel in loc de un rezultat de 7 (dupa cum ne asteptam) sau cel putin o valoare mai mare sau egala cu 4, am obtinut in memorie valoarea 3. Sper tare mult ca ati inteles ce am vrut sa explic aici , deoarece este o treaba foarte interesanta (sau macar mie mi s-a parut ca fiind interesanta).

6. Concluzii

Multe concluzii. Threadurile ne ajuta foarte mult in optimizarea unui process deoarece putem imparti munca in mai multe fire de executie, fiecare fiind independent unul de celalalt. Functionarea threadurilor nu este chiar asa complicata pe cat ma asteptam.

7. Bibliografie

Am pus ceva link-uri in proiect, dar o sa le pun si aici:

https://www.codejava.net/java-se/swing/redirect-standard-output-streams-to-jtextarea
https://www.vogella.com/tutorials/Logging/article.html
Google, Youtube, Suportul de curs, Explicatiile profesorilor, Dezbaterile cu colegii