SISTEM DE PROCESARE A COZILOR

TEHNICI DE PROGRAMARE

UNIVERSITATEA TEHNICA CLUJ-NAPOCA

FACULTATEA DE AUTOMATICA SI CALCULATOARE

SPECIALIZAREA CALCULATOARE SI TEHNOLOGIA INFORMATIEI

Student Urcan Denisa-Teodora

Grupa 30226 CTI-Romana seria B

Cuprins

1. Obiective

1.1. Obiectiv principal

1.2. Obiectiv secundar

2.Analiza problemei

2.1. Analiza problemei

2.2. Modelarea

2.3. Scenarii

2.4. Cazuri de utilizare

3. Proiectare

3.1. Decizii de proiectare

3.2. Diagrame UML

3.3. Structuri de date

3.4. Proiectare clase

3.5. Algoritmi

3.6. Interfata utilizator

4. Implementare

5. Rezultate

6. Concluzii

7. Bibliografie

1. Obiective
   1. Obiectivul principal

Subiectul temei consta in propunerea, proiectarea si implementarea unui sistem de procesare a cozilor in vederea determinarii, analizarii si minimizarii timpului de asteptare. Cerintele aferente acestei teme sunt respectarea paradigmelor programarii orientate pe obiecte, structurate in pachete, clase cu numar maxim 300 linii de cod si metode cu numar maxim 30 de linii. Implementarea temei respecta conventiile de numire Java, si implicit stilul de scriere camel case.

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 in paralel în interiorul aceluiași process, acest process fiind numit multi-threading.

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 mosteneasca 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.

1.2 Obiective secundare

|  |  |  |
| --- | --- | --- |
| Obiectiv secundar | Descriere | Capitol |
| Folosirea structurilor de date convenabile | Structurile de date sunt alese astfel incat sa faciliteze adaugarea si extragerea monoamelor precum si mentinerea unui tip de data sortat. | 3 |
| Impartirea pe pachete | Fiecare pachet este format din mai multe clase care indeplinesc sarcini inrudite, respectand modelul MVC | 3 |
| Impartirea pe clase | Fiecare clasa se ocupa de manipularea unui singur obiect, respectiv a unei singure operatii. | 3 |
| Algoritmi utilizati | Algoritmii utilizati pentru a realiza diferitele operatii sunt cei matematici. | 3 |

1. Analiza problemei
   1. Analiza problemei

Analiza problemei presupune identificarea claselor proiectului, functionalitatile pe care acesta le implementeaza, legaturile dintre ele si modelul impus din punct de vedere al structurii. Programarea orientata pe obiecte ofera strategia de dezvoltare „Top-Down”, prin care utilizatorului ii sunt dezvaluite doar informatiile de suprafata. Aplicatia, care are ca si scop principal efectuarea operatiilor asupra polinoamelor, este structurata in clase, fiecare detinand o functionalitate, implementate ulterior.

Multithreadingul este foarte util pentru executarea mai multor secvente de cod in acelasi timp, altfel spus, un fir de executie poate rula in timp ce altul asteapta un anumit eveniment, spre deosebire de implementarea fara thread-uri, care presupune ca toate operatiile asteapta dupa un eveniment, fie ca sunt corelate, fie ca nu. Astfel, se poate observa o sporire a eficientei executiei programului.

Folosirea tipului de data de tip Coada pentru implementarea aplicatiei permite o implementarea analogica cozilor din lumea reala. O coada este bazata pe principiul FIFO - First In First Out. Astfel, pentru fiecare coada intra si ies clienti, primul care intra este si primul care iese.

2.2 Modelarea

Din punct de vedere al modelarii aplicatiei, s-au implementat diverse clase, fiecare operand asupra unui singur obiect. Clasa Client contine datele specifice fiecarui client : id, timp de sosire, timp de procesare si timp de finalizare, timpul de sosire si cel de procesare fiind calculat random, iar cel de finalizare ca suma dintre timpul de sosire si timpul de procesare la care se adauga ulterior timpul de asteptare la coada aleasa. Clasa Queue contine o coada de clienti care implementeaza metodele necesare adaugarii unui client si stabilirii timpilor specifici, precum si modelarea thread-ului. Clasele View si Controller sunt responsabile de crearea interfetei grafice si comunicarea cu utilizatorul. Clasa Manager modeleaza simularea cozilor, aceasta fiind si ea un thread.

2.3 Scenarii

Utilizarea corecta a aplicatiei presupune introducerea numarului de clienti si selectarea timpului de simulare, in caz contrar fiind afisat un mesaj de eroare. Dupa introducerea datelor se apasa butonul de start si se incepe simularea, iar la final vor putea fi afisate date statistice despre simulare.

Cazul ideal de utilizare presupune un utilizator care doreste sa simuleze procesarea unor cozi dupa cum urmeaza:

* Utilizatorul introduce numarul de clienti in zona destinata, selecteaza intervalul de simulare folosind slide-ul
* Utilizatorul apasa butonul „Start” pentru a porni thread-urile si a incepe simularea
* Programul va simula introducerea, procesarea si iesirea clientilor din fiecare coada.
* La finalul simularii vor fi generate date statistice despre sistemul de cozi, de exemplu: timpul mediu de procesare etc
* Daca se apasa butonul de back, aplicatia va fi resetata si pot fi introduse si simulate date noi de simulare.

Choose simulation interval

Enter the number of clients

Start

view

Statistics

Back

view

2.4 Cazuri de utilizare

1. Utilizatorul apasa butonul de start inaintea introducerii numarului de clienti in zona destinata. Astfel, va fi afisat un mesaj de eroare.

Choose simulation interval

Start

view

ERROR! BAD INPUT

1. Utilizatorul nu introduce un numar in casuta destinata introducerii numarului de clienti, acest lucru fiind de asemenea semnalat.

Choose simulation interval

Start

view

ERROR! BAD INPUT

1. Proiectarea

3.1 Decizii de proiectare

Aplicatia implementata are la baza clasa Client si Queue, precum si un Manager care se ocupa de gestionarea mai multor cozi, distribuite in clasa Queues. Clasa Client detine atributele definitorii, precum ID-ul unui client, timpul de sosire, timpul necesar procesarii precum si timpul la care clientul pleaca, toate acestea fiind reprezentate ca tip de data, implementand metodele care fac vizibile aceste attribute(gettere si settere), constructori specifici, afisarea (toString), precum si implementarea metodei compareTo din interfata Comparable, pentru a sorta ulterior clientii crescator dupa orele sosirii. Clasa Queue, reprezentand o coada, detine o lista de clienti, structurati intr-o structura de priorityBlockingQueue, coada care permite manipularea cu usurinta a concurentei privind threadurile, concomitent clientii fiind introdusi in coada in functie de ora la care au ajuns. De asemenea clasa Queue se ocupa doar de obiectul in cauza, implementand gettere, settere, toString, precum si adaugarea unui client cu modificarea datelor necesare. Pentru o mai mare modularitate, s-a definit o clasa care detine informatiile mai multor cozi, fiind clasa care defineste si strategia aplicatiei, modelata de asemenea din realitate, si anume faptul ca clientul se aseaza la coada cu numar minim de clienti. Cele trei clase prezentate sunt structurate in pachetul Model, impreuna cu clasa Comparing care reprezinta comparatorul (prioritatea) unei cozi. Pachetul controller se ocupa de managementul mai multor cozi, fiind de asemenea reprezentat ca un thread, precum si controller propriu zis, realizand legatura dintre interfata grafica si Model. Vederea constituie cel de-al treilea pachet din cadrul structurarii in modelul classic MVC, implementand interfata grafica.

Controller

Controller

Manager

Controller

View

View

Model

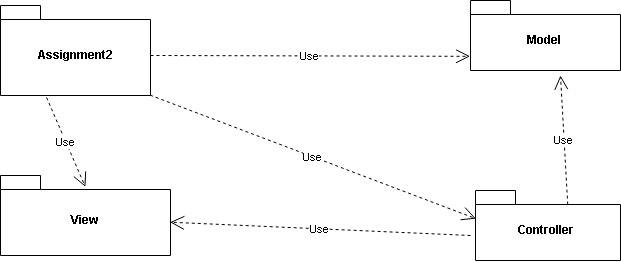
Client

Qeueue

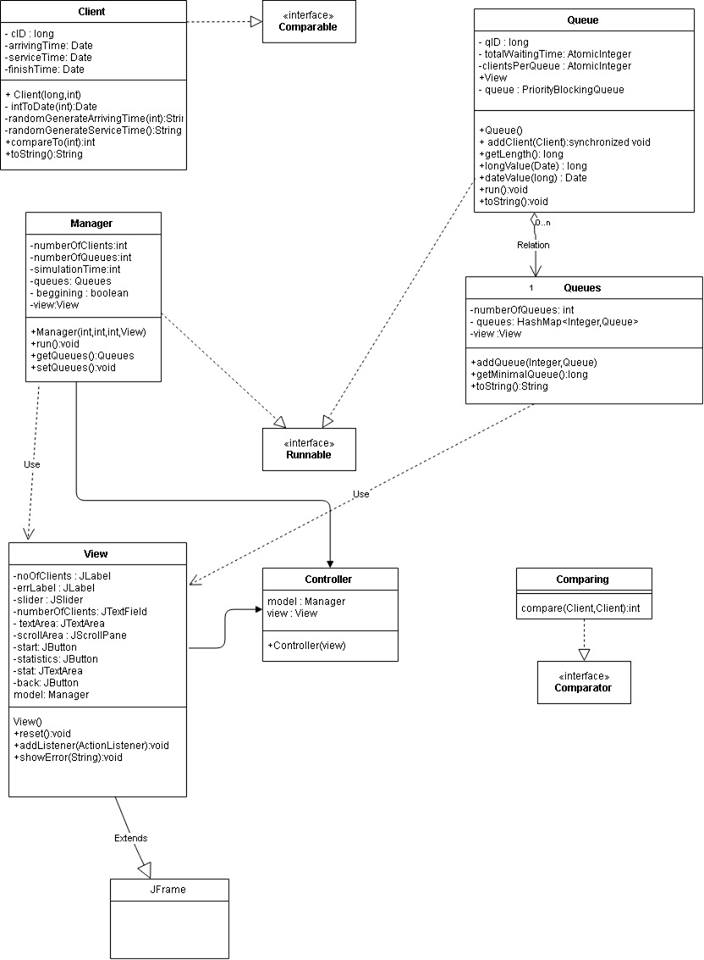
Queues

3.2. Diagrame UML

3.2.1. Diagrama de pachete



3.2.2. Diagrama de clase



3.3. Structuri de date

Pentru stocarea datelor despre clienti s-a folosit o structura de tip PriorityBlockingQueue, structura care faciliteaza operatiile de stergere respectiv adaugare a unui client la coada. Acest tip adauga clienti dupa prioritate, aceasta fiind reprezentata in cazul de fata de timpul de sosire. Totodata, mai multe cozi sunt retinute intr-o structura de tip HashMap, structura care faciliteaza adaugarea unei cozi, fiecare coada fiind reprezentata de cheia HashMap, iar valoarea propriu zisa este reprezentata de o coada. Aceste tipuri de date au lungime variabila, care se modifica odata cu operatia realizata. Totodata, datorita faptului ca adaugarea unui client se face la sfarsitul cozii, parcurgerea listei presupune un timp necesar O(n).

3.4. Proiectare clase

Proiectarea claselor imiplica folosirea unor interfete predefinite in cadrul libariilor Java, precum Comparator implementata de clasa Client pentru a obtine mereu lista de clienti sortati in ordine dupa timpul de sosire, asemenea unei cozi transpuse din realitate. De asemenea, implementarea interfetei ActionListener de catre clasa Controller permite „ascultarea” si interpretarea comenzilor date de utilizator.

3.5. Algoritmi

Algoritmul care se remarca face referire la modul de alegere ideal al cozii, pentru a eficientiza timpul de asteptare.

3.6. Interfata utilizator

Interfata simplista permite folosirea aplicatiei care simuleaza cozile de catre orice utilizator, cu sau fara cunostinte in unul dintre domeniile informaticii.



1. Implementare
   1. Pachetul Model
      1. Clasa Client

Clasa Client defineste cateva atribute definitorii: ID-ul clientului, timpul de sosire, durata de procesare a serviciului sau, timpul de terminare, calculat ulterior. Clasa implementeaza un constructor specific, care primeste ca argument un ID si un interval de simulare. Constructorul genereaza aleator o ora pornind de la ora curenta pana la terminarea intervalului de simulare, date retinute pentru fiecare client. Variabilele instanta alea clasei sunt private din motive de protejare si ascundere a informatiilor legate de implementare fata de utilizator, motiv pentru care clasa contine metode accesoare si mutatoare. De asemenea, clasa Client implementeaza interfata Comparable care permite sortarea in permanenta in ordine crescatoare dupa timpul de sosire (realizata ulterior) prin suprascrierea metodei compareTo(Client). Aceasta clasa se ocupa si de afisarea „prietenoasa” a unui client, pentru a transpune imaginea acestuia cat mai aproape de realitate. Toate tipurile folosite sunt tipuri de data.

* + 1. Clasa Queue

Cozile care urmeaza a fi procesate sunt definite printr-o coada de clienti, structura folosita fiind cea de PriorityBlockingQueue. Cozile propriu zise sunt create in clasa care se ocupa de procesarea mai multor cozi. Coada se instantiaza in constructorul acestei clase, acesta fiind de asemena singurul constructor. Tot din motive de protejare a complexitatiilor interne, variabilele instanta reprezentate de coada clientilor este privata, motiv pentru care si aceasta clasa implementeaza metodele accesoare si mutatoare. Aceasta clasa se ocupa de adaugarea unui client in coada, adaugare simpla care se realizeaza la sfarsit, completata insa de actualizarea timpilor de asteptare pentru fiecare client in parte. Particularitatea care defineste aceasta clasa este folosirea unei cozi sub postura unui Thread, implementand interfata Runnable. Astfel clasa suprascrie metoda run, care preia cate un client din coada, ii determina timpul de asteptare ca fiind diferenta dintre timpul de plecare al clientului dinaintea sa si timpul sau de sosire, la care se aduna timpul de procesare al serviciului dorit de acesta. Metoda actualizeaza timpul de asteptare pentru fiecare client, si se ocupa de stergerea lor din coada la terminarea serviciului.

* + 1. Clasa Queues

Pentru o mai mare modularite, cozile sunt retinute in clasa Queues sub forma unei structuri de tip HashMap, in care cheia reprezinta id-ul cozii, iar cel de-al doilea tip al structurii este o coada. Clasa se ocupa de alegerea strategiei, si anume alegerea cozii cu numar de clienti minim, pentru a simula realitatea.

* 1. Pachetul Controller
     1. Clasa Manager

Clasa Manager, reprezentand un Thread in sine, se ocupa de procesarea simultana a mai multor cozi. Aceasta creeaza propriu zis numarul de cozi in mod aleator intr-un anumit interval, genereaza clientii si pe baza criteriului expus anterior ii distribuie in diferite cozi. De asemenea clasa implementeaza interfata Runnable, thread-urile fiind in executie pe durata timpului simularii.

* + 1. Clasa Controller

Clasa controllerului reprezinta partea clasica a modelului MVC, a carei sarcini este reprezentata de realizarea legaturii intre codul scris si interfata grafica. Aceasta prezinta doua variabile instanta, de tipul View si Model, detinand un constructor care se ocupa de preluarea datelor din vedere si model si adaugarea ascultatorilor vederii, fapt realizat prin compunerea unei clase interioare (innerclass). Desi vederea prezinta mai multe butoane, o metoda de ascultare este deajuns, deoarece se verifica in permanenta sursa ascultatorului si implicit care buton a fost apasat de catre utilizator. Astfel, clasa interioara care defineste si asculatorul butoanelor implementeaza interfata ActionListener, si ii suprascrie metoda actionPerformed(ActionEvent). In cadrul metodei, se trateaza diverse exceptii, precum cazul in care utilizatorul introduce numarul clientilor in format incorect (NumberFormatException), precum si exceptii definite de dezvoltator in sine precum NullInputException, care impune completarea campului care se refera la numarul de clienti. In cazul in care utilizatorul nu introduce numarul de clienti pentru care doreste simularea in format corect, se arunca o exceptie care are ca rezultat afisarea unei erori; de asemenea, neintroducerea unui astfel de camp se semnaleaza cu un mesaj de eroare. Clasa detine si un ascultator asupra sliderului, prin care utlizatorul selecteaza intervalul de simulare , setat initial pe 10(secunde), pentru a evita tratarea exceptiilor in care utilizatorul nu introduce un interval de simulare.

1. Rezultate

Verificarea flow-ului clientilor este realizata in cadrul interfetei grafice, intr-un scrollpanel care permite vizualizarea unui numar mare de clienti.

1. Concluzii

Acest proiect m-a ajutat la intelegerea diferitelor concepte ale programarii orientate pe obiecare, care au fost aprofundate pe parcursul dezvoltarii aplicatiei. Structurarea in pachete si intelegerea nevoi de a structura astfel, crearea unei clase care se ocupa de manipularea unui singur obiect si restrangerea metodelor de rezolvare in mai putin de 30 de linii de cod, respectiv evitarea structurilor imbricate la sugestia laborantului au dus in final la descoperirea unor metode de rezolvare mai simpliste, care pot fi urmarite mai usor. De asemenea, a avut un rol major in intelegera concurentei in programare, prin prisma threadurilor care lucreaza simultan, concent nemaiintalnit.

1. Bibliografie

* https://ubuntuforums.org/showthread.php?t=2222976
* <https://stackoverflow.com/>
* <http://users.utcluj.ro/~igiosan/>
* inf.ucv.ro/documents/tudori/laborator8\_53.pdf
* coned.utcluj.ro/~salomie/PT\_Lic/marcel99