UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

DOCUMENTAȚIE

APLICAȚIE PENTRU GESTIONAREA COZILOR

BUMBU RĂZVAN-VASILE

GRUPA 30223

PROFESOR COORDONATOR: ALEXANDRU RANCEA

CUPRINS

1. OBIECTIVUL TEMEI

Obiectivul principal al acestei teme constă în proiectarea și implementarea unei aplicații care să simuleze un sistem de analiză și procesare a unor cozi, cu scopul de a determina, analiza și minimiza timpul de așteptare.

Cerințele necesare pentru această temă sunt: respectarea paradigmelor programării orientate pe obiect, folosirea firelor de execuție (câte un fir de execuție pentru fiecare coadă), implementarea unei interfețe grafice care să simuleze în timp real evoluția clienților în coadă, generarea în mod aleator a clienților, folosirea unui jurnal pentru a ține evidența datelor pe parcursul simulării.

Pentru obținerea notei maxime, se cere implementarea aplicației pe baza mai multor fire de execuție (utilizarea multi-threading), având pentru fiecare coadă un singur fir de execuție. Altă cerință este extragerea unor date statistice, precum: timpul mediu de așteptare, ora de vârf etc.

2. ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE

Conceptul de fir de execuție (thread) 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, acest proces fiind denumit multi-threading.

Pentru implementarea aplicației folosind firele de execuție, este necesară implementarea interfeței Runnable sau extinderea clasei Thread. Diferența dintre cele două modalități de folosire a firelor de execuție este următoarea: Runnable este o interfață implementată de clasele care o folosesc, acestea fiind capabile să moștenească alte clase, pe cânt Thread este o clasă părinte, care poate fi moștenită, iar clasele care o moștenesc, nu sunt capabile să extindă alte clase.

Analiza problemei presupune identificarea claselor proiectului și a funcționalităților acestuia, precum și legăturile existente între acestea. Așadar, folosind un număr minim de informații, programarea orientată pe obiect ne oferă perspectiva implementării unei aplicații folosindu-ne doar de informații de suprafață.

Multithreading-ul este foarte util pentru executarea în mod simultan a mai multor porțiuni de cod, fiecare sarcină fiind executată în mod independent și rezultatul fiind obținut într-un timp optim. Dacă ne gândim la o implementare fără thread-uri, tot programul nostru reprezintă un singur fir de execuție în care toate operațiile sunt executate în mod secvențial, fiecare așteptând încetarea acțiunilor precedente.

Folosirea structurii de date de tip coadă permite modelarea virtuală a conceptului de coadă din viața cotidiană. Astfel, coada este implementată pe principiul FIFO (first in first out), precum ne putem imagina o coadă la supermarket unde fiecare client își așteaptă rândul să fie servit, iar după ce plătește, următorul client îi va lua locul. Astfel, el fiind primul din coadă, va fi și ultimul plecat. Acest model este implementat în aplicația ce urmează a fi prezentată.

Pentru modelarea problemei am folosit principiile programării orientate pe obiect: alegerea structurilor de date astfel încât acestea să faciliteze operațiile de adăugare, de extragere și de sortare a datelor; împărțirea pe pachete (organizarea claselor înrudite în pachete pentru menținerea lizibilității aplicației); împărțirea pe clase (fiecare clasă manipulează un singur obiect).

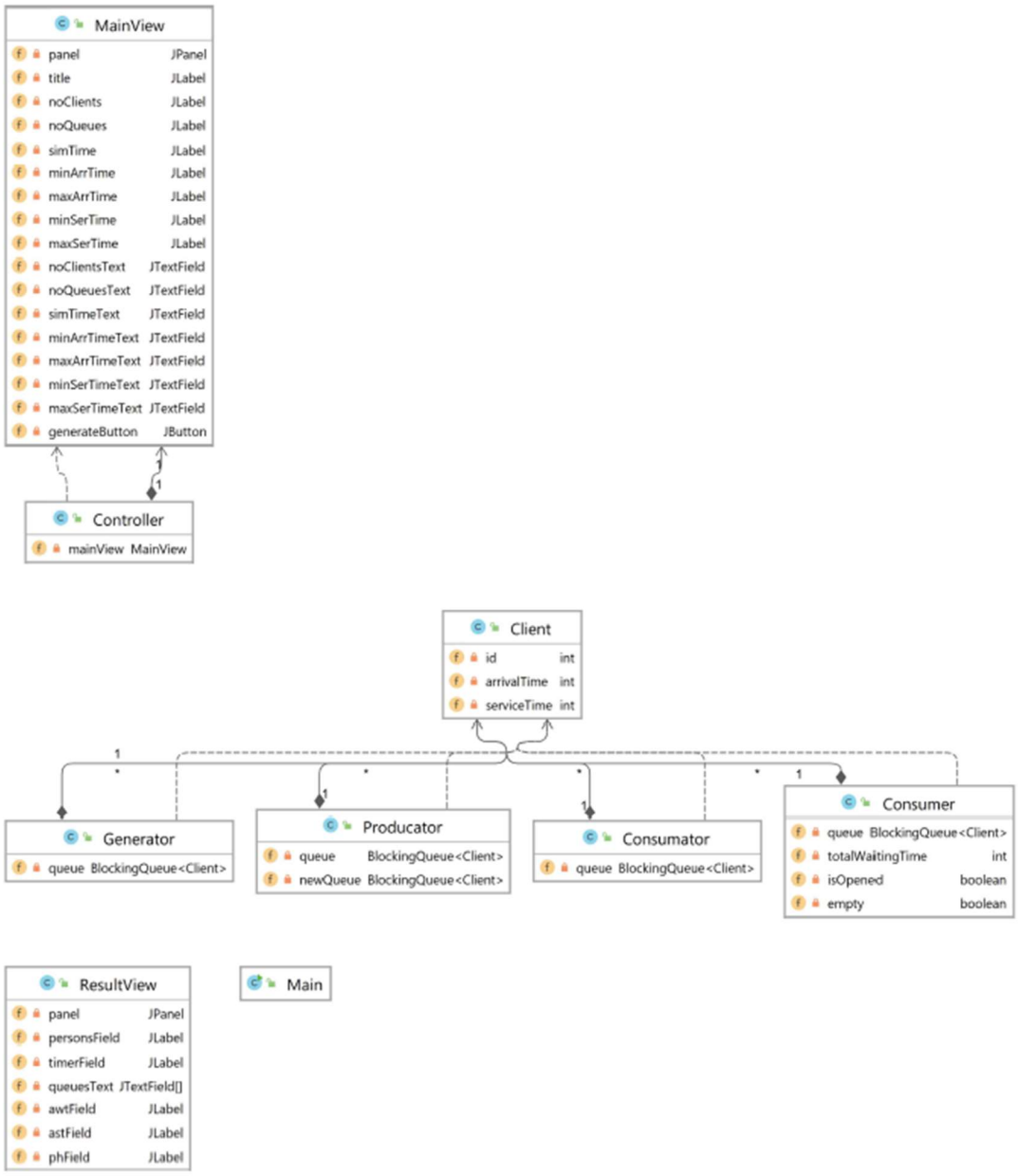
Pentru ca aplicația să poată fi utilizată în mod corect, trebuie să fie urmărite următoarele condiții:

- Trebuie să se introducă numărul de clienți
- Trebuie să se introducă numărul de cozi
- Trebuie să se introducă timpul maxim de simulare
- Trebuie să se introducă timpul minim de așteptare
- Trebuie să se introducă timpul maxim de așteptare
- Trebuie ca timpul minim de așteptare să fie mai mic sau egal cu timpul maxim de așteptare
- Trebuie să se introducă timpul minim de servire
- Trebuie să se introducă timpul maxim de servire
- Trebuie ca timpul minim de servire să fie mai mic sau egal cu timpul maxim de servire
- Trebuie să se de click pe butonul GENERATE QUEUES doar după ce utilizatorul a introdus toate datele în mod corect

3. PROIECTARE

Unified Modeling Language (UML) este un limbaj standard pentru descrierea de modele și specificații pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.

Diagrama UML a calculatorului este formată din 6 clase: Client, Consumer, Controller, Generator, Main, MainView și ResultView. Legăturile dintre aceste clase sunt prezentate în imaginea de mai jos.



Model-View-Controller (MVC) este un model arhitectural utilizat în ingineria software. Succesul modelului se datorează izolării logicii de business față de considerentele interfeței cu utilizatorul, rezultând o aplicație unde aspectul vizual și nivelele inferioare ale regulilor de business sunt mai ușor de modificat, fără a afecta alte nivele. In cazul temei mele, clasa Model este înlocuită de clasa Generator. Clasa View este alcătuită din două clase difertite (MainView și ResultView), întrucât vom avea două fereste (fereasta principală și fereastra de rezultate).

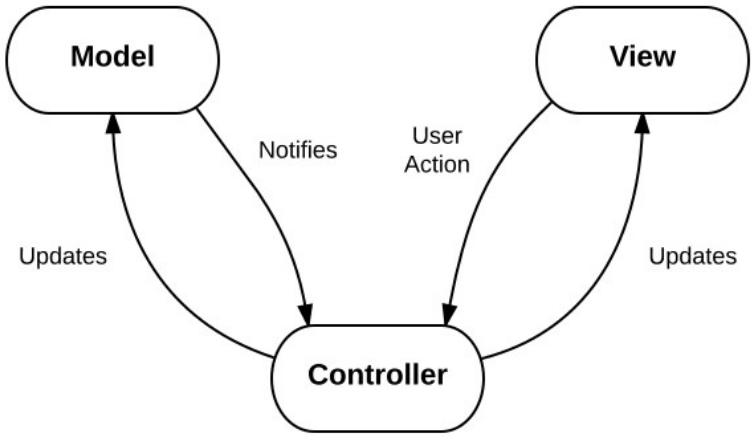
Simulatorul de cozi are la bază o arhitectură de tip MVC. Astfel, clasa Controller devine principala componentă. Aceasta realizează conexiunile dintre clasele Generator, MainView și

ResultView. Modelul este în permanență actualizat prin intermediul controllerului, căruia îi transmite datele. Totodată, view-ul este actualizat în permanență de controller, căruia îi transmite informații despre acțiunile utilizatorului.

Modelul controlează operațiile de utilizare a informației (trimise dinainte de către rangul său superior) pentru a rezulta o formă mai ușoară de înțeles.

View-ul corespunde reprezentării grafice, sau mai bine zis, exprimării formei datelor. Acesta este reprezentat de interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la Controller.

Controllerul are rolul de a controla întreaga aplicație. Se pot controla fișiere, scripturi, programe (în general cam orice tip de informație permisă de interfață). În acest fel putem diversifica conținutul nostru de o formă dinamică și statică, în același timp.



4. IMPLEMENTARE

- **Clasa Client**

În cadrul acestei clase sunt create obiectele de tip client, care vor fi puse mai târziu în cozi. Fiecare client are ca și atribute; un id, timpul de sosire și timpul de servire.

Metodele implementate în cadrul acestei clase sunt:

- Constructorul – pentru a crea obiectele de tip Client;
- public int compareTo (Client client) – compară timpurile de sosire ale clienților;
- gettere și settere.

- **Clasa Consumer**

În cadrul acestei clase sunt generate și gestionate firele de lucru ale cozilor. Fiecare obiect de tip Consumer are ca și atribure: timpul total de așteptare, variabila de tip boolean isOpened care probează să vadă dacă coada de clienți este deschisă sau nu și variabila de tip boolean empty care probează să vadă dacă coada de clienți conține sau nu elemente.

Metodele implementate în cadrul acestei clase sunt:

- Constructorul – pentru a crea obiectele de tip Consumer;
- public void updateWaitingTime ( ) – calculează timpul total necesar pentru epuizarea persoanelor dintr-o coadă;
- public void joinQueue (Client client) – pentru adăugarea unui nou client în coadă;
- metoda suprascrisa run ( ) – aceasta se ocupă de gestionarea thread-urilor pe cozile mici (cozile unde sunt trimiși clienții când le vine rândul);
- gettere și settere;

- **Clasa Controller**

În cadrul acestei clase este utilizată fereastra de pornire de tip MainView.

Metodele implementate în cadrul acestei clase sunt:

- Constructorul – pentru a crea obiectul de tip Controller;
- Clasa ButtonListener – aceasta implementează un ActionListener care răspunde la apăsarea butonului de generare a cozilor. De la această acțiune va porni întreaga execuție a programului. Este generată coada mare, sunt create cozile mici și este creată noua fereastră de afișare a rezultatelor.

- **Clasa Generator**

În cadrul acestei clase se petrec majoritatea evenimentelor. Această clasă este cea mai importantă clasă a aplicației, întrucât aceasta implementează majoritatea metodelor. Atributele acestei clase sunt: noQueues, noClients, simulationTime, minArrivalTime, maxArrivalTime, minServiceTime, clients (inițială de clienți), queues(lista de cozi), runningTime, averageWaitingTime, averageServingTime, peakHour.

Metodele implementate în cadrul acestei clase sunt:

- Constructorul – pentru crearea obiectului de tio Generator;
- public void clientsGenerator ( ) – pentru generarea random a clienților pe baza specificatiilor;
- public void queuesGenerator ( ) – pentru generarea cozilor goale pe baza specificațiilor;
- public void allQueuesAreEmpty ( ) – care verifică dacă toate cozile sunt goale sau nu;

- public int getShortestQueue ( ) – care returnează coada cu cel mai mic timp de așteptare;
- public void startQueues ( ) – care pornește execuția cozilor;
- public void stopQueues ( ) – care oprește execuția cozilor;
- public void setSize (int sizes) – care setează dimensiunea cozilor;
- public void consolePrint (int sizes) – folosită pentru a printa în timp real datele în consolă;
- public String filePrint (String s, int sizes) – folosită pentru a genera șirul de caractere care va fi scris în fișier;
- public static void write (String s, File f) – folosită pentru a scris șirul de caractere în fișier;
- public int getPeakHour (int peakHour) – care returnează momentul în care cozile sunt cele mai aglomerate;
- public float getAverageServingTime (float averageServingTime) – care returnează media timpilor de așteptare pentru a fi serviți;
- metoda suprascrisa run ( ) – această metodă se ocupă de gestionarea cozii mari de clienți, dar și de alte acțiuni precum crearea ferestrei de rezultate și actualizarea acesteia, afișarea datelor în consolă, scrierea datelor în fișierul text etc.

- **Clasa MainView**

În cadrul acestei clase sunt construite obiectele de tip MainView (fereastra de pornire a programului).

Metodele implementate în cadrul acestei clase sunt:

- Constructorul – care creează obiectul de tip fereastră principală;
- gettere și settere.

- **Clasa ResultView**

În cadrul acestei clase sunt construite obiectele de tip ResultView (fereastra de rezultate a programului).

Metodele implementate în cadrul acestei clase sunt:

- Constructorul – care creează obiectul de tip fereastră de rezultate;
- public void update ( ) – care realizează actualizarea permanentă a ferestrei de rezultate;
- gettere și settere.

- **Clasa Main**

  În cadrul acestei clase sunt construite obiectele de tip MainView și Controller.

  Metodele implementate în cadrul acestei clase sunt:

➢ public static void main (String args) – această metodă este metoda principală prin intermediul căreia programul va rula;

5. REZULTATE

Ca și rezultate, am obținut o simulare fluentă a procesului de intrare și ieșire din cozi, în funcție de datele de simulare furnizate. Optimizarea timpilor de așteptare este una foarte bună. Fiecare client ajunge să petreacă un timp cât mai scurt la coada.

6. CONCLUZII

În concluzie, această temă m-a ajutat să aprofundez paradigmele Programării Orientate pe Obiect. Astfel,am reușit să îmi dezvolt abilitățile de a lucra cu liste, de a lucra cu interfața grafică și totodată de a mă familiariza cu structura de tip Model View Controller (MVC). De asemenea, consider că am reușit să îmi îmbunătățesc și abilitățile gândirii logice și matematice. Cel mai important lucru pe care am reușit să-l învăț de la această tema a fost utilizarea firelor de lucru (Threads). Astfel, de acum mă simt capabil să creez aplicații mai complexe ale căror fire de lucru trebuie să se execute în paralel, programul fiind astfel optimizat. Totodată am înțeles necesitatea utilizării unui sistem de gestiune a cozilor în viața de zi cu zi.

Ca și dezvoltare ulterioară, aplicația ar putea fi înfrumusețată. Interfața grafică ar putea primi un aspect mai modern, ar putea afișa siluete care să simuleze persoanele care se află la momentul respectiv în coadă etc.

7. BIBLIOGRAFIE

- https://www.baeldung.com/
- https://ro.wikipedia.org/wiki/Pagina_principal%C4%83
- https://www.w3schools.com/java/
- https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html
- https://www.youtube.com/watch?v=r_MbozD32eo&ab_channel=CodingwithJohn
- https://www.youtube.com/watch?v=d3xb1Nj88pw&ab_channel=JakobJenkov