Documentatie

Tema 2 : Simulare cozi clienti

Student: Radu Vlad

Grupa 30226

Materie: Tehnici de programare

Profesor indrumator: Mitrea Dan

Cuprins

1. Obiectivul temei .
2. Analiza problemei, modelare, scenarii, cazuri de utilizare .
3. Implementare .
   1. Diagrame UML .
   2. Clase si metode .
   3. GUI .
4. Rezultate si Testare .
5. Concluzii .
6. Bibliografie .
7. Obiectivul temei .

Se doreste simularea si implementare unei aplicatii cu Q cozi si un numar de N taskuri, reprezentand un magazin cu case de marcat si clienti. Simularea functionarii magazinului se face avand o strategie bazata atat pe cel mai scurt timp de asteptare cat si pe cea mai scurta coada de la o casa.

Coada trebuie sa gestioneze clientii luand pe rand cate un client dupa strategia FIFO (first in first out) sa proceseze clientul si sa il elimine din coada dupa ce a fost procesat.

Vizualizare acestor cozi se face atat printr-un fisier cat si printr-o interfata grafica care afiseaza in timp real datele procesate in aplicatie ilustrand continutul fiecarei cozi. In interiorul fisierului se va scrie fiecare pas pe care il parcurgem in aplicatie valorile din fiecare coada si timpul corespunzator pasului.

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

In cadrul aplicatiei se generaza o lista de clienti pe care dorim sa ii trimitem la casele de marcat. Pentru ca simularea sa se desfasoare in mod real, adica casele sa functioneze simultan se folosesc threaduri pe fiecare casa creeata. Punerea clientilor in cozi se realizeaza in functie de una dintre cele doua strategii prezentate: Daca avem un client C sunt doua optiuni prin care se adauga intr-o coada

* In coada ce are waiting time cel mai mic, se adauga noul client C
* In coada care are cei mai putini clienti in asteptare, se adauga noul client C

In cadrul aplicatie am implementat ambele metode, selectia strategiei se realizeaza schimband valoarea atributului selection policy printr-un enum , totodata am ales ca in cadrul simularii sa ma folosesc doar de strategia Time Strategy.

Text

Description automatically generated

Text

Description automatically generated

Fiecare dintre cele doua clase de Strategii vor trebui sa implementeze metoda din Interfata numita Strategy. Ideea pe care se bazeaza Time Strategy este de a gasi timpul minim de asteptare si coada la care se refera acesta paracurgand toata lista de cozi. La final taskul este adaugat in coada si se efectueaza operatile specific problemei.

Strategia Shortest Queue este identica ca idee, se gaseste coada care are dimensiunea minima, adica numarul minim de client in coada si coada la care se refera dimensiune dupa care adaugam taskul.

Pentru a genera lista de clienti cu care lucreaza casele de marcat se foloseste metoda GenerateNRandomTasks(..) in cadrul careia fiecare task primeste un id, un timp arrivalTime si un serviceTime. Valorile se genereaza din doua intervale deja stabilite avand capetele maracate de datele de intrare ale problemei.

Text

Description automatically generated

Casele lucreaza cu lista de clienti sortata crescator dupa timpul de venire (arrival time) pentru aceasta se folosete metoda sort din Collections, astfel ca clasa Task v-a implementa metoda compareTo() cu prioritate la arrivalTime.

Un mod usor de vizulizare modului in care functioneaza aplicatia si capabilitatilor acesteia este prin desenarea unei diagrame Use Case astfel:

Diagram

Description automatically generated

1. Implementare .
2. Diagrama UML

Unified Modeling Language (UML) joaca un rol important in dezvoltarea de software, dar si pentru sistemele non-software din numeroase domenii, deoarece este o modalitate prin care putem vizualiza comportamentul si structura unui sistem sau proces. UML ajuta la prezentarea posibilelor erori din structurile aplicatiilor, comportamentele sistemelor si alte procese.

Diagrama clasei este elementul principal al modelarii orientate pe obiecte.

A screenshot of a computer

Description automatically generated with medium confidence

1. Clase si metode .

Pentru structurarea codului am utilizat modelul MVC (Model, View, Controller), adică am împărțit codul în 3 pachete, la care am adăugat clasa Main.

Clasa Main controlează tot MVC-ul, având două instanțe; una a clasei View, pe care o setăm vizibilă, pentru a putea vedea interfața grafică (GUI) și cealaltă este o instanță a clasei Controller, care leagă Model-ul de View, fiind principalul punct de comandă al celor două.

Fără această clasă, care are rol unificator, utilizatorul nu s-ar putea bucura nici de interfața grafică cu utilizatorul (GUI), și nici de funcționalitățile simulării unei cozi. Noi vedem în timp real introducerea, procesarea și excluderea clienților din cozi și vedem acest lucru pe fiecare coadă în parte.

Text

Description automatically generated

Pachetul model continue 2 clase Server si Task

Clasa Task, contine 3 atribute de tipul private ID (id ul de identificare al taskului), arrivalTime (= time de sosire ), serviceTime (= timp de asteptare ) si setterele si getterle necesare pentru a obtine datele. Totodata clasa Task implementeaza interfata Comparable si metoda compareTo() prin care ne permitem sa sortam lista de taskuri, crescator dupa arrival time. Pe lângă acestă metodă, în clasă suprascriem și metoda toString, care ne ajută să afișăm într-un anumit format clienții

Text

Description automatically generated

Clasa Server, functioneaza ca si un thread deoarece implementeaza interfata Runnable. Aceasta interfata vine cu metoda run() ce trebuie implementata. Metoda run a acestei clase trebuie sa preia cate un element din coada si sa il proceseze. Astfel: ii scade timpul de servire, verfica daca timpul este egal cu 0 in caz ca trebuie sa il elimine din coada. In acelasi timp decrementeaza si waiting period cu 1 si opreste threadul pentru o perioada de o secunda.

Text

Description automatically generated

Ca atribute clasa Server continue un BlockingQueue<Task> adica este o coada ce are o implementare care blocheaza firul in timpul operatiilor de inserare si stergere. Aceasta inseamna ca blocheaza firul atunci cand coada si a atins capacitatea maxima in timpul operatiei de insearare. In mod similar, blocheaza firul atunci cand coada este goala in timpul operatiei de stergere. Variabila prin care îi setăm lungimaea este reprezentată de numărul maxim de clienți pe care îi avem în total

Pe langa blocking queue, server contine un atribut de tipul atomic integer waitingPeriod ce reprezinta timpul de asteptare pentru coada respective.

Atomic în Java este un concept foarte important într-un [multithreading](https://www.tutorialcup.com/ro/Java/multithreading-java.htm) mediu inconjurator. Este unul dintre utilitarele simultane care asigură că mai multe fire utilizează eficient resursele partajate fără a duce la probleme. Acest lucru se datorează faptului că aceste resurse partajate își pot schimba valorile în timpul execuției într-un mediu cu mai multe fire. Putem implementa atomicitatea în clase, operații și variabile.

Text

Description automatically generated

Pe lângă toate acestea, avem pentru fiecare variabilă instanță getter și setter, plus o suprascriere a metodei toString pentru a afișa o coadă și elementele din ea.

Text

Description automatically generated

Pachetul business logic continue clasele Scheduler, SimulationManager, ShortestQueueStrategy, TimeStrategy ultimele doua sunt folosite cu interfata Strategy folosind selectia din enumul SelectionPolicy

Clasa Scheduler:

Această clasă conține un ArrayList de cozi (serversList), un ArrayList de threaduri (threadsList) un număr maxim de cozi maxNoServers și un numar maxim de clienți pentru fiecare coadă (maxTasksPerServer). Ultimele două variabile instanță sunt oferite ca parametru constructorului.

Totodata contine si un atribut startegy de tipul interfetei Strategy prin care folosim strategia aplicata in simulare.

Text

Description automatically generated

In constructor se intializeaza atributele si se contruiesc listele pentru servers si threads.

In aceasta clasa se implementeaza si o metoda care schimba intre cele doua strategii. In functie de abordare pe care vrem sa o ia problema noastra.

Text

Description automatically generated

Restul codului din acestă clasă este reprezentat de getter-e și setter-e si de o metoda startThreads prin care este parcursa si pornita lista de threaduri.

Interfata Strategy contine capul de metoda pe care trebuie sa il implementeze fiecare dintre clasele ShortestQueueStrategy si TimeStrategy in fuctie de selectia data prin enumul Selection Policy.

Text

Description automatically generated

Se parcurge lista de serveri, se cauta serverul care are timpul cel mai mic de asteptare. In lista acestui server este adaugat Taskul t primit ca parametru in lista metodei. Totodata timpul de asteptare este actulizat si incrementat cu valoarea service timeului lui t.

Clasa ShortestQueueStrategy are o abordare identica cu cea a lui TimeStrategy ambele metode vor adauga taskul t in serverul optim.

Text

Description automatically generated

Clasa simulation Manager este cea mai complexa clasa, prin intermediul clasei Scheduler aceasta gestioneaza lista de serveri si lista de threaduri. Constructorul acestei clase primeste ca parametrii toate valorile extrase din interfata grafica, tot aici este initializat obiectul de tip Scheduler, se instantiaza un frame de output in care va fii afisat rezultatul simularii si se generaza o lista random de taskuri pentru serveri.

Text

Description automatically generated

Clasa implementează și interfața Runnable, ceea ce înseamnă că suprascrie metoda run(). Această metodă se ocupă cu logica de la fiecare secundă a timpului curent. Atât timp cât nu am depășit limita de timp, se extrage din lista de clienți un client și se adaugă într-o coadă pe baza strategiei de timp alese prin SelectionPolicy. Se actualizează frameul de output și stringul rezultat ce urmeaza sa se scrie in fișier. Crește timpul curent și se oprește simularea dacă atât cozile, cât și listele de clientți sunt goale.

Text

Description automatically generated

Text

Description automatically generated

Tot in clasa constructor am scris doua metode ajutatoare writeFunction si writeInFile pentru a putea afisa rezultatele obtinute.

Functia writeInFile in generaza un nou obiect de tip fiser in care va scrie rezultatul obtinut in stringul rezultat.

Text

Description automatically generated

1. **GUI**

In pachetul gui am scris clasele Controller, SimulationFrameInput, SimulationFrameOutput.

GUI este reprezentată de Graphical User Interface, un termen folosit nu numai în [Java,](https://ro.eferrit.com/ce-este-java/) ci și în toate limbajele de programare care suportă dezvoltarea de GUI-uri. Interfața grafică de utilizator a unui program prezintă un afișaj vizual ușor de utilizat pentru utilizator. Acesta este alcătuit din componente grafice (de exemplu butoane, etichete, ferestre) prin care utilizatorul poate interacționa cu pagina sau [aplicația](https://ro.eferrit.com/exemplu-cod-java-pentru-a-construi-o-aplicatie-simpla-gui/) .

Un GUI include o serie de elemente de interfață cu utilizatorul, comenzile de intrare, cum ar fi butoanele, listele derulante, casetele de selectare și câmpurile de text.

Interfața grafică cu utilizatorul este creată prin clasa Simulation Frame Input și legată de restul codului prin Controller.

Clasa Controller contine un constructor care initializeaza view ul. Tot in clasa Constructor este implementata metoda de tip ActionPerformed prin care sunt extrase toate valorile din interfata grafica, se cosntruieste managerul si un thread principal cu acesta.

Text

Description automatically generated

Clasa SimulationFrameInput extinde JFrame. JFrame este o versiune extinsă a clasei Frame care adaugă suport pentru un comportament de desenare special. Adiţional, JFrame permite componentelor Swing MenuBars să fie ataşate nu numai în partea de sus a ferestrei dar oriunde în fereastră. Toate obiectele asociate unui JFrame sunt manipulate de o instanţă a clasei JRootPane, care este singura componentă-fiu a unei instanţe JFrame.

Clasa SimulationFrameInput contine toate Labelurile si TextFieldurile necasare construirii intefetei grafice. Are un constructor prin care sunt initializate si adaugate in Jpaneluri atributele clasei si mesajele necesare.

Prin intermediul interfetei grafice utilizatorul este capabil sa interactioneze cu aplicatia si sa vada modificarile efectuate intr-un mod mult mai comfortabil. Totodata interfata grafica ofera posibilitatea de afisare dinamica a datelor de iesire.

La executia programului se deschide un frame input frame in care utilizatorul va introduce datele necesare simularii aceste date au anumite restrictii pe care trebuie sa le respectam.

Restrictile sunt adresate campurilor arrivalTime si serviceTime acestea trebuie sa apartine unor intervale prestabilite in documentatia temei.

Dupa ce datele au fost introduse se apasa butonul validate button care va porni simularea ruland threadul principal si celelalte componenete. In momentul executiei se genereaza o fereastara de iesire in cadrul careia sunt afisate rezultatele obtinute cu comentarii cu rol ajutator. De asemenea in folderul aplicatiei se genereaza un fiser myFile.txt care va contine inforamtile afisate si in simulation output,

Graphical user interface, text, application

Description automatically generated

1. Rezultate si Testare

Testarea codului se realizeaza atat in frameul de output cat si intr-un fisier test.

Pentru frame ul de output am ales o startegie dinamica, rezultatele sunt afisate doar la timpul curent frameul se actualizeaza la fiecare iteratie a timpului. In schimb in fisierul text datele vor fi scrie secvential.

1. Concluzii

In urma implementarii acestui proiect am realizat imporatnta functionalitatilor pe care acesta le are in domeniul comertului in magazine si supermaketuri cat si in domeniul industriei fiind capabil sa fie integrat in procedee mult mai complexe de concurenta a taskurilor ce trebuie sa fie rezolvate.

Pe langa asta, personal, proiectul a reprezentat o sumedenie de avantaje in dezvoltarea stilului meu de programeare. M a ajutat sa imi amintesc anumite concepte de programare in java si respectarea anumitor paradigme de programre.

Cu siguranta proiectului i se pot aduce imbunatatiri prin rezolvarea confictelor de sincronizare si imbunatatire strategiior de adaugare a taskurilor in cozi.

6. Bibliografie:

<https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf>

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/>