Logo, company name

Description automatically generated

QUEUE SIMULATOR

Documentație

Pentru: Tehnici de Programare

Student: Luncian Lidia, grupa 30227

Cuprins:

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie
8. Obiectivul temei

Obiectivul acestei teme a fost acela de a realiza o aplicație cu o intefață „User Friendly” pentru un simulator de cozi implementate cu ajutorul thread-urilor, ce analizeaza timpul de asteptare si de procesare al comenzilor unor clienti. Fiecare coada contine un singur thread, acestea sunt sincronizate si lucreaza in paralel. Se foloseste o interfata grafica prin intermediul careia utilizatorul introduce de la tastatura datele simularii: numarul de clienti ce trebuie procesati, numarul de cozi disponibile, timpul maxim de simulare, intervalul de timp in care pot ajunge clientii la coada, si intervalul de procesare al fiecarei comenzi.

Această aplicație este proiectată conform paradigmelor de programare orientată pe obiecte, folosind limbajul de programare Java si modelul MVC. Au fost create 3 fisiere ce prezinta rezultatele a 3 teste.

1. Analiza problemei

În această secțiune voi prezenta modul de interacționare al utilizatorului cu interfața grafică. Această interfață este intuitivă, ușor de folosit oricine ar fi utilizatorul. Vor fi prezentate 2 scenarii de utilizare a aplicatie.

Mai jos este prezentat use-case-ul acestei teme.

Diagram

Description automatically generated

Userul reprezinta actorul/ utilizatorul acestei aplicatii. Prima interfata cu care utilizatorul intra in contact este Simulator. Aceasta contine campuri ce trebuie completate de user pentru a porni simularea. Ceea ce trebuie sa introduca utiliazatorul este specificat prin etichetele de dinaintea campurilor de completat. Daca nu introduce suficiente date si apasa butonul „Start” va aparea mesajul „Introduceti date valide”. Daca datele introduse sunt valide si se va apasa butonul „Start” se va porni interfata ce prezinta starea fiecareicozi la fiecare moment de timp, va prezenta evolutia cozilor in timp real. Cand simularea se incheie, in locul etichetei ce prezinta momentul de timp, va aparea mesajul „The end”, si asteapta ca utilizatorul sa inchida simulator din butonul de inchidere. Nu se inchide automat si nici nu afiseaza rezultate, precum timpul de asteptare mediu, sau ora de varf.

Această interfață arată astfel:

Graphical user interface

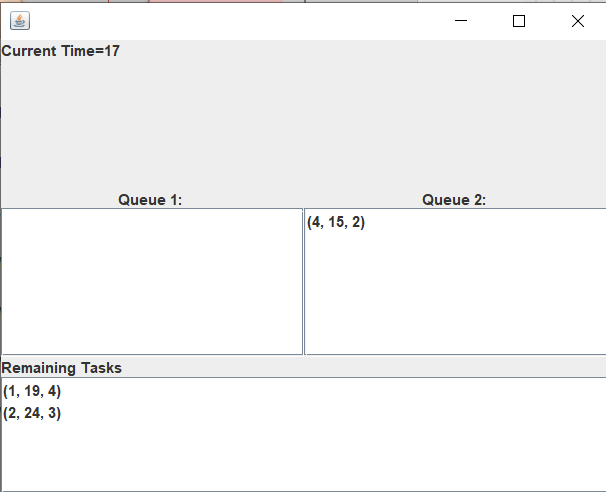
Description automatically generated

Mesajul care apare daca nu au fost introduse suficiente date:

Graphical user interface, application, Word

Description automatically generated

Interfata ce prezinta evolutia cozilor in timp real:



Simulatorul trebuie sa permita userului sa introduca datele necesare simularii, sa poata porni simularea si sa vizualizeze rezultatele in timp real. Simularea se face implicit folosind strategia shortest\_time.

Scenariul de utilizare:

1. User-ul nu introduce suficiente date
2. Acesta apasa butonul start
3. Apare mesajul de eroare „Introduceti date valide”

Scenariul 2:

1. Utilizatorul introduce numarul de clienti
2. User-ul introduce numarul de cozi totale
3. User-ul introduce timpul de simulare
4. User-ul introduce limita inferioara a intervalului de sosire al clientului la una din cozi
5. User-ul introduce limita superioara a intervalului de sosire a clientului la una din cozi
6. User-ul introduce limita inferioara a intervalului de procesare a unei comenzi
7. User-ul introduce limita superioara a intervalului de procesare a unei comenzi.
8. Utilizatorul apasa butonul „Start” al aplicatiei
9. Se porneste interfata de simulare ce prezinta in timp real modul de asezare al fiecarui client la una din cozi, si modul de procesare al comenzilor acestora.
10. Cand apare mesajul „The end” in locul etichetei ce prezinta timpul curent, utilizatorul poate apasa butonul X si iese din aplicatie.

Fiecare client este generat random alegand random un timp de sosirre la coada si un timp de procesare al comenzii din cele doua intervale date de utilizator.

In imaginea de mai sus ce prezinta simularea au fost introduse datele: numar clienti: 4, numar cozi: 2, timp de simulare: 60s, interval de sosire: [2,30], interval de procesare: [2,4]. Captura de ecran prezinta momentul 17 al simularii, unul din task-uri s-a incheiat deja, iar pentru cel de-al doilea a inceput deja procesarea.

1. Proiectare

Am urmarit implementarea acestei teme folosind limbajul Java si modelul de proiectare orientat pe obiect MVC (Model – View – Controller), fiind un model interactiv si usor de implementat.

Nu am impartit proiectul in pachete, toate clasele se afla in pachetul default al proiectului. Exista o clasa Model: SimulationManager, doua clase View: View si SimulationView si o clasa Controller, ce respecta modelul MVC, restul claselor: Task, Server, Strategy, Scheduler, FileWritter, ConcreteStrategyTime si ConcreteStrategyQueue tin de model, de implementarea proiectului.

Diagrama UML:

Diagram

Description automatically generated

1. Implementare

In aceasta sectiune voi prezenta fiecare clasa. Intrucat aplicatia nu este impartita pe pachete, voi prezenta fiecare clasa dupa categoria din acre fac parte: model, view sau controller.

**„ConcreteStrategyQueue”**

**Clasa ConcreteStartegyQueue.** Aceasta clasa implementeaza strategia shortest queue de alegere a cozii in care urmeaza sa fie plasat un task. Se va folosi ca si coada optima cea in care se afla cei mai putini clienti.

Text

Description automatically generated

**„ConcreteStartegyTime”**

**Clasa ConcreteStrategyTime.** Aceasta clasa implementeaza strategia shortest time de alegere a cozii in care urmeaza sa fie plasat un task. Se va cauta coada cu cel mai mic timp de asteptare si se va adauga clientul la aceasta coada. Desi ambele strategii sunt implementate, in acest proiect utilizatorul nu poate alege strategia de asezare a clientilor la cozi, ci este aleasa implicit ca fiind shortest time.

Text

Description automatically generated

**„Strategy”**

**Interfata Strategy.** Pentru a generaliza metoda addTask, aceasta a fost declarata in interfata Strategy si implementata de cele doua clsa ementionate mai sus: ConcreteStrategyQueue si ConcreteStrategyTime.

Text

Description automatically generated

**„SelectionPolicy”**

**Enum-ul SelectionPolicy.** Acest enum va ajuta in alegerea politicii de simulare. Avem optiunea de a alege dintre shortest\_queue (coada cu cei mai putini clienti, cea mai scurta), si shortest\_time (coada cu cel mai scurt timp de asteptare), pentru a determina la care coada se va aseza clientul din coada de asteptare.

**„Task”**

**Clasa Task.** Aceasta clasa reprezinta clientul care va fi asignat unei cozi in functie de timpul de procesare cel mai scurt al cozii. Clientul se caracteriaeaza prin idC = id-ul clientului, arrivalTime = momentul in care este gata sa ase aseze la o coada, processingTime = cat timp dureaza ca comanda sa sa fie procesata si waitingTime = timpul de asteptare incepand cu momentul in care e gata sa se aseze la o coada si momentul in care incepe sa ii fie procesata comanda. Acest waitingTime il vom folosi pentru a calcula timpul mediu de asteptare.

In aceasta clasa am implementat constructorul care creeaza si initializeaza un client, functiile de get si set pentru fiecare caracteristica a clientului. In plus am suprascris metoda toString, functie folosita pentru afisarea clientilor generati. Aceasta afiseaza id-ul, timpul la care intra intr-o coada, si timpul de procesare cuprinse intre paranteze.

Aceasta clasa implementeaza si Comparable si este implemeantata functia compare, care va compara timpul de intrare in coada, si va fi utilizata pentru sortarea clientilor odata generati in functie de arrivalTime.

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

**„Server”**

**Clasa Server.** Aceasta clasa implementeaza structura propriu-zisa a cozii, a serverului, implementand interfata Runnable. Astfel fiecare coada va fi de fapt un thread. Toate cozile vor rula in acelasi timp, deci thread-urile vor lucra in paralel.

Fiecare coada este caracterizata printr-o coada de task-uri, un waitingPeriod de tipul AtomicInteger care retine timpul de asteptare, un id care retine numarul cozii, un flag de tip boolean, runnable, care semnalizeaza daca thread-ul inca ruleaza sau nu, si o variabila in care va fi calculata perioada medie de asteptare a fiecarei cozi.

Pe langa constructor, gettere si settere au fost implementate si metodele: addTask, stop, run, getTasksList, decrementWaitingPeriod, incrementWaitingPeriod si toString.

In aceasta clasa este implementata metoda addTask care va adauga clienti la coada si va adauga la variabila waitingPeriod timpul de servire al clientilor din coada. Metoda stop seteaza runnable pe fals. Cu ajutorul aceste metode se va opri thread-ul cozii cand va fi cazul.

Metoda run este esentiala, aceasta face ca fiecare server sa doarma atunci cand nu exista clienti in coada si sa astepte procesarea comenzii atunci cand este un client in coada ca mai apoi sa elimine clientul a carui timp de procesare s-a incheiat, si sa incepa procesarea urmatorului client daca mai exista, si tot asa atat timp cat coada este activa si nu a fost apelata metoda stop pentru ea.

Metoda decrementWaitingPeriod decrementeaza valoarea variabilei waitingPeriod.

Metoda to string afiseaza umarul cozii si elementele care se afla in ea. Daca este goala afiseaza closed.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**„Scheduler”**

**Clasa Scheduler.** Cu ajutorul acestei clase task-urile sunt asignate cozilor. Aceasta clasa contine o lista de servere folosita la adaugarea de clienti in functie de strategia aleasa. Inserarea unui client la o coada se va face folosind un obiect de tipul Strategy prin metoda dispatchTask.

In aceasta clasa sunt implementate si metodele: changeStrategy care ofera posibilitatea schimbarii strategiei de a pune clienti in coada, stopServers care va opri serverele cand toate cozile sunt goale, serversAreDone, verifica daca toate cozile sunt goale, updateProcessingTime care incrementeaza timpul de asteptare a cozii si decrementeaza timpul de procesare al comenzii in curs, decrementWaitingTime, decrementeaza timpul de asteptare al tuturor cozilor, si incrementWaitAverage. Acestea vor fi apelate in SimulationManager, in metoda run.

Text

Description automatically generated

Text

Description automatically generated

**„SimulationManager”**

**Clasa SimulationManager.** Aceasta clasa este clasa Model din MVC cu care vom putea porni simulare si thread-urile si va controla evolutia cozilor. Aceasta este compusa din datele care vor fi introduse de utilizator in interfata, contine o variabila de tip Scheduler care va fi responsabila cu managementul cozilor si cu distribuirea clientilor, variabile ce vor crea fisierele ce vor contine pasii simularii si rezultatele obtinute si o variabila ce va crea fereastra de simulare in timp real. Tot aici se va afla si lista de cozi si lista de clienti ce vor fi generati aleator folosind metoda generateNRandomTasks.

Metoda getTasks returneaza toate task-urile in functie de serverul dib=n care fac parte sub forma unui vector de liste. Metoda generateNRandomTasks se foloseste de datele introduse de utilizator in intrafata si va genera clienti cu id unic si cu date generte aleator din intervalele date. Acestia vor fi sortati in functie de timpul de intrare in coada. Tot in aceasta metoda se calculeaza si timpul mediu de processare al comenzilor.

Text

Description automatically generated

Metoda principala run contine o bucla while care atata timp cat timpul curent nu este mai mare decat timpul imita dat de utilizator simularea sa continue. Scheduler-ul va primi cate un client, daca momentul lui de sosire coincide cu momentul curent se va scoate din lista de asteptare si va fi asignat unei cozi. Apoi se face verificarea cu urmatorul client. Daca nu mai sunt clienti care pot fi adaugati in coada la acel moment, se trece mai departe, se consemneaza in fisier acest moment, apoi se calculeaza numarul de clienti din fiecare coada. Se verifica daca una din cozi are mai multi clienti decat in momentul precedent de timp si se actualizeaza variabila peak hour. Se afiseaza momentul si pe interfata de simulare. Se incrementeaza timpul curent al simularii si se actualizeaza timpii de asteptare al cozilor si al clientilor.

Daca nu mai sunt clienti in lista de asteptare si nici in cozi, iar timpul curent este inca mai mic decat timpul limita atunci serverele se opresc si se iese din simulare. Cand se iese din while se opresc serverele si se afiseaza peakHour si timpul mediu de asteptare in fisiere.

Tot aceasta clasa contine si metoda main care porneste aplicatia.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**„FileWritter”**

**Clasa FileWritte.** In aceasta clasa se creaza metodele de scriere in fisiere.

**„View” „SimulationView”**

**Clasele View si SimulationView.** Aceste clase implementeaza interfetele cu utilizatorul. SimulationView contine o metoda displayQueues ce afiseaza in timp real evolutia cozilor. Aceasta metoda este apelata in functia run din clasa SimulationManager.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**„Controller”**

**Clasa Controller.** Creeaza legatura intre datele din interfata si model. Implementeaza action listener-ul pentru butonul de start.

1. Rezultate

Am reușit să bifez toate subpunctele temei. Am generat clientii random, am folosit multi-threading, un thread pentru fiecare coada. Am creat o interfata atat pentru introducerea datelor, cat si pentru afisarea in timp real a evolutiei cozilor. Toate datele simularii sunt generate si in fisiere de tip text, iar la finalul acestora apar si rezultatele simularii, timpul mediu de asteptare si peak hour. Aceste rezultate nu sunt afisate si in interfata. Am generat cele 3 fisiere cu cele trei seturi de date date.

Pentru testul 1 cu datele:

Text

Description automatically generated

Au fost generat fisierul Test\_1.txt cu datele urmatoare:

Text, table

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated with medium confidence

Pentru testul 2, cu datele:

Text, letter

Description automatically generated

A fost generat fisierul Test\_2.txt:

A picture containing calendar

Description automatically generated

Graphical user interface, text, table

Description automatically generated with medium confidence

Si testul 3, cu datele:

Text, letter

Description automatically generated

A fost generat fisierul Test\_3.txt

Background pattern

Description automatically generated

A picture containing text

Description automatically generated

1. Concluzii

În urma realizării acestui proiect am învățat cum sa folosesc thread-urile in aplicatie, cum sa generez o interfata real-time display. Am invata mai bine sa lucrez cu fisiere si cu variabile de tip AtomicInteger.

Aplicatia poate fi bineinteles imbunatatita. Se pot face verificari, daca intevalele au fost scrise corect, sa nu fie timpul maxim mai mic decat cel minim. Sa verifice daca a introdus numere si nu alt fel de caractere. Se poate adauga sa aleaga strategia pe care vrea sa o adopte si sa afiseze tot in interfata si rezultatee, nu doar in fisier.

1. Bibliografie
2. <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
3. <http://www.tutorialspoint.com/java/util/timer_schedule_period.htm>
4. <http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-andthreadpoolexecutor.html>
5. [Java Generate Random Number Between Two Given Values - Stack Overflow](https://stackoverflow.com/questions/5271598/java-generate-random-number-between-two-given-values)