DOCUMENTATIE

*TEMA 2*

Contents

[DOCUMENTATIE 1](#_Toc100130349)

[*TEMA 2* 1](#_Toc100130350)

[1. Obiectivul temei 2](#_Toc100130351)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 2](#_Toc100130352)

[3. Proiectare 2](#_Toc100130353)

[4. Implementare 3](#_Toc100130354)

[5. Rezultate 5](#_Toc100130355)

[6. Concluzii 5](#_Toc100130356)

[7. Bibliografie 5](#_Toc100130357)

# Obiectivul temei

Obiectivul principal al temei este realizarea unei aplicatii de management pentru asignarea clientiilor unor cozi. Obiectivele principale sunt reprezentate prin lucrul cu threaduri intr-o maniera concurenta , avand in vedere aspecte importantente precum proprietatiile acid ( Atomicitate Aceasta problema presupune realizarea unor clase client ,server , simulation management , si scheduler.

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Problema presupune ca utilizatorul sa introduca in cadrul interfetei datele necesare unei simularii corecte precum numarul de clienti , de cozi, timpul maxim de simulare , intervale de sosire si service precum si strategia utilizata . In cazul in care datele introduse sunt incorecte se va afisa o fereasta de eroare avand mesajul “Eroare date incorecte” pentru a semnala utilizatorul de aceste greseli. Metoda de rezolvare a problemei presupune o implementare producer – consumer unde producatorul va trimite la cozi clientii generati ,iar cozile vor procesa in paralel clientii primiti.

# Proiectare

Proiectarea se bazeaza pe multithreading si implementarea producer – consumer. In momentul porniri programului utilizatorul va trebui sa introduca datele necesare simularii.Dupa inserarea datelor utilizatorul va apasa butonul procesare si simulare. Datele vor fi verificate si daca corecte se va deschide o noua fereasta GUI unde se va putea observa in timp real progresarea simularii . Dupa incheierea simularii , daca timpul curent atinge timpul maxim precizat anterio de catre utilizator threadurile vor fi oprite fortat chiar daca acetea nu ai incheiat procesarea intregilor clientii si se vor afisa datele average\_waiting\_time si average\_service\_time. In acest moment utilizatorul are optiunea de a repeta simularea anterioara cu aceleasi date insa rezultate diferite sa inchida fereastra resprectiva si sa introduca alte date pentru o noua simulare . Fiecare metoda multithreading implementeaza interfata Runnable avand metoda run . Aceasta metoda este mai eficienta deoarece putem implementa un numar mare de interfete fata de a extinde o singura clasa Thread , insa dejavantajul este ca nu avem acces direct la toate metodele Thread.

Programul java este alcatuit din 7 clase : 2 clase pentru interfata si 5 pentru multithreading. Clasa client inglobeaza 3 atribute : id , service\_time si arrival\_time . Aceasta clasa sta la baza programului deorece reprezinta clientii din cadrul managementului de taskuri. Clasa server sau coada contine o multitudine de atribute si reprezinta locul in care clientii pleaca din coada mare de asteptare si sunt procesati . Clasa scheduler distribuie clientii in functie de strategia aleasa si de asemenea opreste si porneste threadurile de cozi pentru ca simularea sa aiba loc intro anumita ordine. Clasa Simulator\_management este responsabila cu coada originala de clienti , acestia sunt generati aici si contorizeaza timpul total de simularea al aplicatiei. Clasele de interfata sunt responsabile cu a oferii utilizatorului o metoda interactiva de a lucra cu elementele simularii. Clasele prezinta doar implementarea interfetelor Runnable : server si simulator\_manager cat interfata comparable in cadrul clasei client . Nu este prezenta o relatie de mostenire intre clasele proiectului. In main se initializeza clasa process\_frame care la randul sau initializeaza un obiect simulator\_manager ce in continuare initializeaza celelalte componente ale simularii .

O importanta parte a implementari o reprezinta metodele wait si notifyall ce pot fi apelate cu ajutorul oricarui obiect . Metoda wait va pune serverele in asteptare pana cand simulator\_manager va apela metoda notifyall pentru a reporni threadurile oprite mai devreme . Aceste metode au nevoie de un bloc syncronized pentru a obiectul ce este transmis si la fel in cadrul tuturor threadurilor sa stie cine este ownerul care apeleaza metoda wait la un anumit timp. Un important aspect al implementari este faptul ca se regaseste de foarte multe ori metoda try and catch pentru a se evidentia locul unde pot aparea eventuale erori datorita metodelor sensibile pecum wait , sleep , notifyall , write to file .

# Implementare

Clasa Client este o clasa generica ce contine 3 atribute : Id , service\_time si arrival\_time. Aceasta clasa ofera doar metode de setter si getter precum si ToString si Compareato ( Utilizata pentru a sorta clientii dupa timpul de sosire.In metoda toInteface clientul este atribuit cu un emoji pentru a fi mai usoara vizualizarea in cadrul interfetei fata de fisierul logs unde clientul este atribuit cu stringul "(" +Id +","+arrival\_time +","+ service\_time+ ") ".Clasa contine settere si gettere pentru atribute.

Strategy este o enumeratie formata din 2 elemente ShortestQueue si ShortestTime cu ajutorul carea la fiecare moment de timp se alege un ce coada vor fi trimisi clientii. Pentru ShortestQueue clienti vor fi trimisi in coada cu numarul cel mai mic de clienti cea ce ofera o implementare mult mai fair fata de celelate cozi . Shortest time reprezinta coada unde numarul service\_time dat de timpul total de service generat de toti clienti din coada de clienti este minim , acesta optiune nu asigura o implementare foarte fair din moment ce la o coada se pot afle un numar foarte mare de clienti cu service timeul minim iar la alte cozi un numar foarte mic de clientii dar cu un service time foarte mare.

Clasa Server este reprezentata de locul unde clientii sunt procesatii in maniera concurenta. Atributele nr\_Clients si service\_time sunt utilzate pentru a alege serverul de livrat clienti in cadrul strategiei respective.Atributul running este utilizat pentru a oprii fortat serverul chiar daca nu au fost procesati toti clienti. Writer este untilizat impreuna cu metoda write\_to\_file pentru a construii fisierul logs al unei simularii . Atributul lock este un obiect dummy utilzat pentru ca un thread sa intre in strea de pauza sau waiting pentru a mentine o ordine intre server si simulator\_manager. Metoda run incepe cu un scurt wait pentru ca Threadurile sunt pornite inaintera managerului .Daca un client nu se afla la caserie sau current acesta este luat din coada de clienti din cadrul serverului. Daca un client se afla deja in current acesta va avea service time decrementat , iar daca atinge zero acesta este inclocuit de urmatorul client din clients.Syncronized lock pune serverul intr-o stare de asteptare pana cand este notificat de catre scheduler.Iar acest proces este repetat pana la icheirera simularii. Clasa contine settere si gettere pentru atribute. Metoda interface atribuite lui current o locomotiva emoji pentru a fi mai usor de vizualizat iar celelalte sub forma de vagoane . Pentru metoda write\_to\_file se scrie in fisier prin append starea actuala a serverului prin Qeueu id ( un numar unic generat la initializarea lui scheduler din cadrul simulator\_manager ) clienti sub forma to string.

Clasa schedulre consista dintr – un numar scazrut de atribute.Metoda wake\_all este folosita pentru a notifica serverele.Metoda stop\_all opreste fortat toate serverele din rulare. Metoda send\_cliente timite din coada principala de clienti in functie de strategie se gasete serverul minim din cadrul listei de servere si apeleaza addCilent pentru serverul gasit. La initializare acesta primeste bufferul file pentru servere in cadrul caruia se va scrie logs , numarul de servere de initializat si strategia pe care o va aplica de trimitere a clientiilor .

Clasa simulator\_Manager prezinta componenta pricincipala din cadrul simulatii.Aceasta este alcatuita dintr-o gama foarte mare de atribute .Acesta este clasa principala a aplicatiei intrucat contine scheduler ce la randul sau va avea acces la serverele pe care le va crea si deasemenea interfata de simulare pe care o actualizeaza periodic , bufferul filewriter este creat aicit deoarece acesta are nevoie sa scrie in fisier cat si serverele create la care este trimis mai departe prin intermediul obiectului scheduler. Primele date sunt dateele initiale si esentiale ale simularii , writer este utilzat pentru a scrie in fisierul logs clienti ramasi asteptati ( coada mare originala ) si timpul curent al simularii.Metoda generate este utilzata pentru a genera un blockingqueue( utilizata pentru a forta o accesare secventiala dint partea serverelor ) avand un numar stabiit de clienti precum si intervalele de variatie a atributelor unui client , dupa care colectia este sortata crescator in functie de arrival time . Metodele print , toInterface , write\_to\_file sunt strict utilizate pentru a afisa utilizatorului fie in interfata grafica si in fisierul logs datele in timp real ale simularii.Metoda run este activa cat timp timul actual al simularii este mai mic decat maximul de timp de simulare introdus de utilizator.Daca timpul de arrival al mai multa clienti coincide atunci cat timp acesti timpi sunt egali clienti sunt trimisi catre servere prin sheduler.Se actualizeaza si se scrie in fisierul logs clientii precum si interfata grafica. Se incrementreaza timpul curent si se notifica serverele pentru asi relua procesarea cat timp se gasesc clientii urmatori de trimis.La finalul simularii prin sheduler vom putea forta tote servelele sa afisam timpi average calculati .In cadrul acestei metode se apeleaza sleep de 2 ori unul din ele proportional cu numarul de cozi pentru ca acestea sa aiba timp sa isi incheie mai intai procesarea clientului actual si apoi sa intre in starea de wait , insa va incetini considerabil procesul de simulare. Metoda toInterface trimite catre interfata coada de clienti sub forma de locomotive . Metoda state creaza un buffer din servere pentru a putea fi afisate asupra interfetei . Clasa contine gettere si settere pentru atributele sale . In constructor se intializateaza obiectul cu datele primite din process\_frame , apoi se creaza interfata simulation\_frame si este setata visible , precum schedule este intializat .

Clasa Process\_frame este utilizata pentru a prelua datele simulari de la utilizatori si ale procesa.In interiorul acestei clase se instrantiaza si porneste threadul de simulator\_manager.Dupa inserarea datelor in cadrul text\_fieldurilor marcate prin labelori utilzatorul apasa pe butonul procesare si simulare . Daca datele introduse au fost corecte se va porni simularea si se va afisa o noua fereastra gui , in caz contrar se va afisa o fereasta care atentioneaza utilzatoril de greselile cauzate.

Clasa Simulation\_frame este reprezentate printr - o fereastra gui ce afiseaza starea actuala a serverelor precum si timpul curent si coada de clienti in asteptare, la fiecare moment de timp din simulator\_manager aceste frame se va actualiza prin intermediul metodelor update cu ajutorul datelor din simulator\_manager.La finalizarea simularii se vor afisa datele de timp average .Dupa acest moment utilizatorul are optiunea de a putea porni o noua simulare cu datele introduse anterior si sa primeasca rezultate diferite sau sa inchida fereasta si sa introduca date noi corecte pentru a pornti o noua simulare .Acesta clasa contine doar metode de update al starii serverelor si afisarea acestorar print shimbarea regulata a datelor din campurile text . Metoda de update preia textfieldurile din cardul frameului si le suprascrie la fiecare moment de timp astfel creanduse iluzia unui actualizari in real time. Frameul prezinta scrollable frame in interiorul sau astfel incat utilizatorul sa poata sa vizualizeze fara probleme intreaga stare a simularii. Daca o coada nu contine niciun client la registru ( current ) aceasta va fi afisata cu mesajul closed.

# Rezultate

Testarea este alcatuita din 3 cazuri diferite formate din date de intrare diferite . Acestea vor produce de fieacare data rezultate diferite precum este prezentat in fisierul logs. Fisierul este scris simulan de catre servere prin write\_to\_file , si pentru a nu exista ciocniri acesta metoda este syncronized , si de asemenea de catre simulator\_manager.

# Concluzii

In cadrul acestei teme se aprofundeaza notiunea de thread si lucrul cu acesta in cadrul multithreading . Se introduc concepte noi precum Atomic ce ofera atomitatea si consistenta datelor atunci cand mai multe threduri acceseaza in acelasi moment de timp aceeasi variabila si se creaaza inconsistente. Syncronized utilizat pentru a supune thredurile sub o oarecare ordine secventiala prin faptul ca la un anumit moment de timp doar un thread poate sa apeleza acea metoda. BlockingQueue ce ofera atomicitate si consistenta a datelor in cadrul accesarii multiple .

# Bibliografie

<https://dsrl.eu/courses/pt/>;

<http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>;

<https://www.tutorialspoint.com/java/util/timer_schedule_period.htm>;

<https://www.baeldung.com/java-wait-notify>;

https://www.baeldung.com/java-synchronized;