Universitatea Tehnica din Cluj-Napoca

Facultatea de Automatica si Calculatoare

Queue management system

Elev: Rosca Ion-Sergiu

Grupa:30225

Cuprins

1.Obiectivul temei

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

3.Proiectare

4.Implementare

5.Rezultate

6.Concluzii

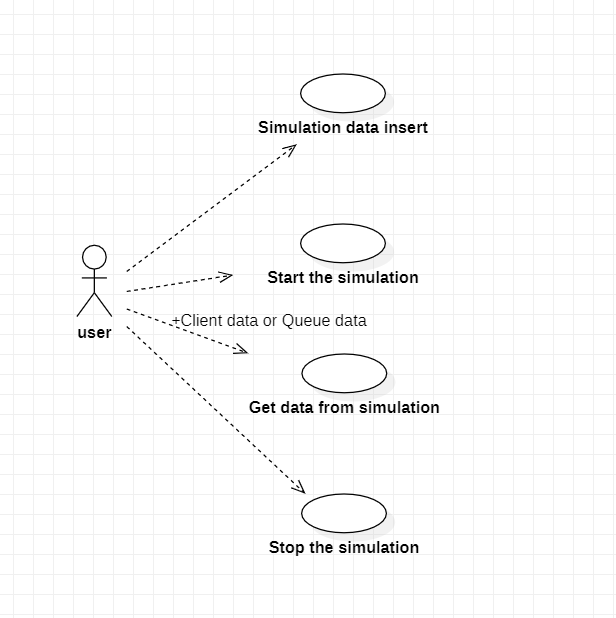
7.Bibliografie

1.Obiectivul temei

Se cere proiectarea si implementarea unui algoritm pentru plasarea clientilor la cozi intr-un mod eficient in functie de doua criterii. Aceste doua criterii sunt: SHORTEST TIME sau SHORTEST QUEUE, adica plasarea clientilor in functie de timp (cel mai mic) sau plasarea clientilor in functie de dimensiunea cozii(cea mai scurta). Clientii sunt reprezentati prin setul de date: timp minim sosire, timp maxim sosire, timp minim de procesare, timp maxim de procesare. Procesul de plasare este definit prin urmatorii parametrii: numar de clienti, numar maxim de cozi, timpul de simulare, si politica(criteriul) dupa care sunt plasati clientii(SHORTEST\_QUEUE sau SHORTEST\_TIME). Clientii sunt generati aleator in intervalele de timp(timpul minim de sosire si timpul maxim de sosire). Plasarea clientilor se face dupa timpul de sosire, clientii cu timpul de sosire mai mic avand prioritate. Ca sa pot realiza acest lucru am ordonat lista cu clientii generati aleator crescator in funcite de timpul de sosire crescator. Pentru inceperea procesului trebuie completate toate campurile si apoi apasarea butonului Start. Necompletarea campurilor corespunzator va duce la aparitia de erori.

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

Datorita faptului ca pot fi mai multe cozi, adica mai multe procese care au loc simultan, am folosit Thread -uri pentru implementarea acestui program, acestea avand caracteristica de Multi Thread-ing, adica procese care pot avea loc simultan. Am considerat fiecare coada ca fiind un Thread asa ca am implementat clasa pentru coada (Queue) cu implements Runnable deoarece am creat un Thread cu aceasta. Am adaugat ca si caracteristica pentru fiecare coada timpul de asteptare pentru a putea calcula ulterior si timpul mediu de asteptare si pentru a putea adauga clientii in functie de timpul de asteptarea pentru fiecare coada. Clientii au fost introdusi folosind o metoda din clasa implementata pentru coada in funcite de politica selectata. Clientii au fost adaugati cu functia deja existena add dupa care am adugat timpul de procesare al clientului la timpul de asteptare al cozii in care a fost introdus clientul. Pentru adaugarea clientilor in functie de cele doua politici am implemetat doua clase care implementeaza metoda din interfata in care se gaseste metoda respectiva pentru a o suprascrie, astfel incat se va folosi metoda dorita de fiecare data. Pentru implementarea in functie de timp am parcurs lista de cozi (List<Queue>) si am comparat fiecare timp cu o variabila, care initial are o valoare foarte mare, imposibil de egalat. Dupa ce am atribuit acelei valori timpul de asteptare al celei mai mici cozi am parcurs inca o data lista de cozi si am verificat care lista are acelasi timp cu cel din variabila in care am salvat cel mai mic timp. In acea stiva am introdus clientul cu ajutorul metodei implementate in clasa pentru coada. Pentru plasarea in functie de dimensiunea cozii am procedat similar, diferenta fiind ca in variabila pentru a salva valoarea cea mai mica am salvat dimensiunea cea mai mica din lista de cozi.

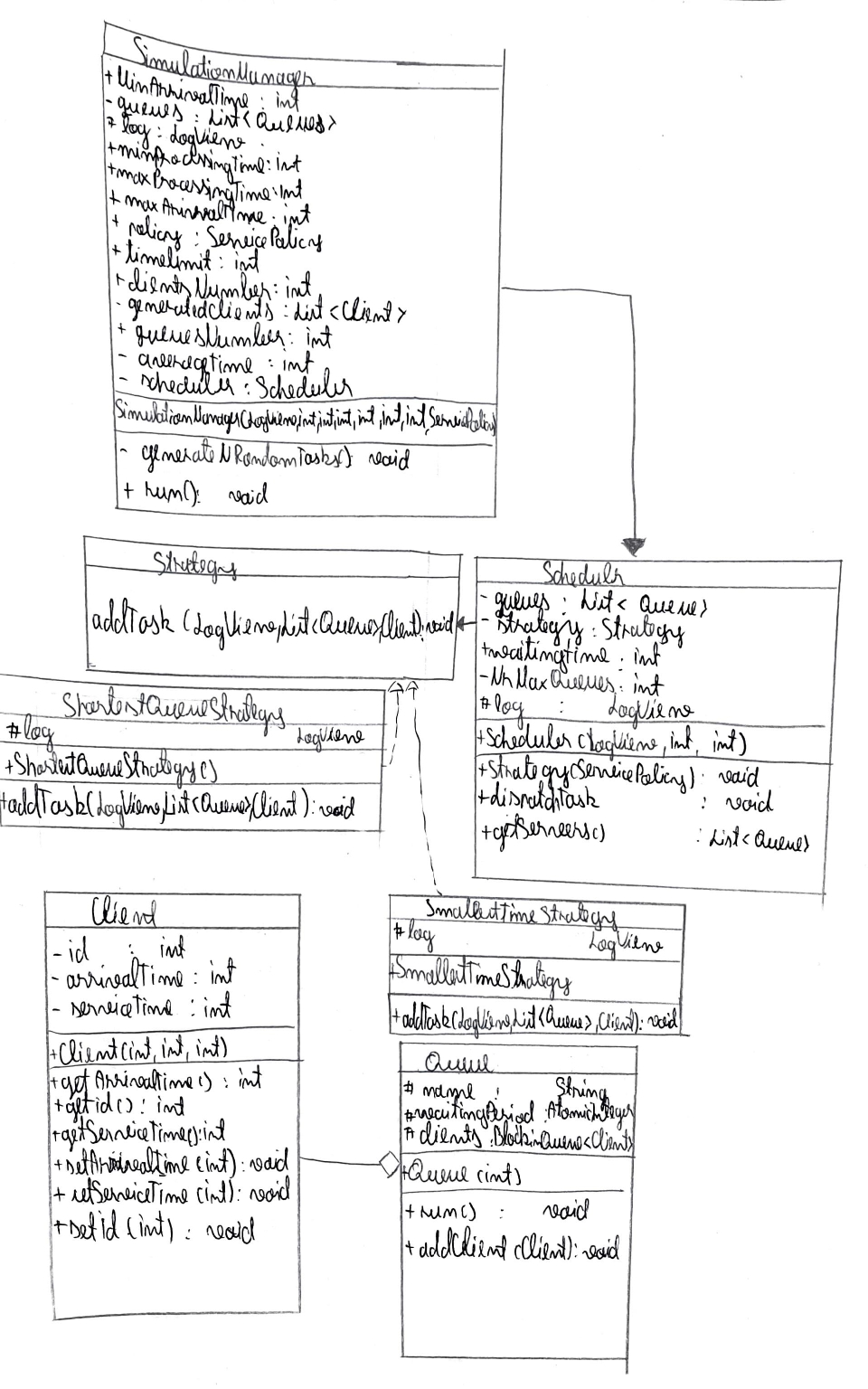


3.Proiectare

Pentru o implementare modulara si pentru o parcurgere mai usoara a proiectului m-am folosit de structura MVC (Model-View-Controller). In View am pus cele doua clase pentru interfata, una fiind cea in care introducem datele, iar cealalta fiind cea pentru Log, acolo putem vedea in timp real ce clienti si in care coada se plaseaza fiecare. Aici vedem si timpul simularii, dar si timpul mediu de asteptare acestea actualizandu-se in timp real. In interfata pentru introducerea datelor vom gasi campuri unde putem introduce toate datele de care Clientii si procesul au nevoie. Butonul start din aceasta fereastra porneste procesul. Pentru obtinerea rezultatului dorit este necesara introducerea corespunzatoare a tuturor campurilor disponibile.

In Model am introdus toate Clasele necesare implementarii „motorului” programului. Aici am implementat clasa pentru client, coada, sortare, cele doua modalitati de sortare(in functie de cel mai mic timp sau cea ce mai scurta coada), si scheduler -ul care reprezinta clasa in care am organizat impartirea task urilor si generarea Thread- urilor pentru fiecare coada. Enum-ul ServicePolicy reprezinta cele doua politici pentru modul de organizare: SHORTEST\_TIME, SHORTEST\_QUEUE. In interfata Strategy am definit metoda care este implementata in cele doua clase pentru distrubuirea clientilor in funcite de una din cele doua politici, astfel ca se va suprascrie metoda si va fi folosita doar cea dorita de fiecare data. In clasa pentru client am definit variabilele si a implementat setter si getter pentru fiecare, astfel incat sa putem accesa atributele private Clasei pentru client si in alte clase. Toate aceste metode din clase sunt folosite in clasa pentru simulare in care am implementat simularea programului. In aceasta clasa am generat clienti aleator, i- am introdus intr- o lista si am ordontat- o crescator in functie de timpul de sosire. Am inceput cu timpul de la 0 iar apoi l am crescut treptat si l-am comparat de fiecare data cu timpul de sosire al fiecarui client. In cazul in care timpii coincid se vor distribui clientii cu timpul de sosire egal cu timpul curent. Dupa incrementarea timpului vom face ca Thread- ul sa astepte o secunda.

In Controller am inclus View ul si metoda de simulare din Model. Aici am adaugat actiune pentru butonul de start astfel incat la apasarea acestuia ase vor salva valorile din campurile din interfata in niste variabile locale pe care le am introdus in parametrii functiei de simulare. Am inceput un thread nou cu aceasta functie deoarece si l- am pornit cu ajutorul metodei start.



4.Implementare

Pentru inceput, in clasa pentru client am declarat atributele si am implementat setter si getter pentru fiecare dintre acesta, astfel pot accesa acele atribute. Fiecare client reprezinta un task, iar fiecare coada reprezinta un server care proceseaza acel task. In coada am declarat timpul de asteptare, numele(pentru a putea afisa in ce coada este plasat fiecare client) si o lista de clienti, in care o sa inseram clientii cu ajutorul metodei addClient. Metoda pentru adaugarea clientilor consta in adaugarea in lista cu ajutorul metodei add si adaugarea la timpul de asteptare, timpul de procesare al clientului inserat in coada. In functia run din coada am oprit Thread- ul cu timpul de procesare \* 1000 deoarece timpul trebuie introdus in milisecunde. Dupa acest lucru vom elimina clientii din stiva. In clasa Scheduler creez numarul maxim de cozi, care este unul dintre atributele clasei. Pentru fiecare coada creez un Thread nou pe care il asociez cozii, dupa care pornesc Thread- ul. Cu ajutorul metodei Strategy care are ca parametru strategia introdusa din interfata, unei variabile i se va atribui una din cele coua metode, in functie de politica introdusa ca parametru, astfel incat metoda addTask se va supra scrie. Metoda dispatch task apeleaza metoda addTask aceasta fiind suprascrisa de metoda implementata inainte. Pentru a returna lista cozilor am implemetat functia getServers care returneaza lista cozilor printr- un return. Pentru a evita eventualele erori am impelementat si un enum ServicePolicy in care am adaugat cle doua critetii pentru plasarea clientilor: SHORTEST\_TIME ( in functie de cel mai mic timp) , SHORTEST\_QUEUE ( in functie de cea mai scurta coada ) . In clasa ShortesQueueStrategy am implementat modul de distribuire al clientilor in funcite de timp. Pentru a realiza acest lucru am parcurs lista cozilor si am comparat dimensiunea fiecarei cozi cu o variabila care initial avea o valoare foarte mare, imposibil de egalat. In cazul in care dimensiunea unei cozi era mai mica decat valoarea din acea variabile voi salba in acea variabila dimensiunea cozii. Dupa parcurgerea listei in acea variabila voi avea cea mai mica dimensiune din lista a cozilor. Mai parcurg odata o lista iar in momentul in care gasim o coada cu dimensiunea egala cu valoarea din acea variabila introducem clientul pe acea coada dupa care am oprit programul cu fuctia break. In clasa SmallestTimeStrategy distribuim clientii in functie de timpul minim de asteptare pentru fiecare coada. Pentru a realiza acest lucru am parcurs lista si am salvat intr – o variabila cel mai mic timp de asteptare al cozilor din lista de cozi. Dupa ce am salvat intr- o variablia cea mai mica dimensiune a unei cozi din lista de cozi am mai parcurs odata lista si am verificat ce lista este cea cu dimenisunea cea mai mica. Dupa ce am gasit acea lista inseram clientul si iesim din bucla ( break ). Aceste doua functii de organizare a clientilor implementeaza metoda addTask din interfata, astfel se suprascrie acea metoda iar clientii vor fi repartizati dupa modul in care utiliazatorul doreste sa ii repartizeze de fiecare data. Pentru a pune la un loc toate aceste clase am implementat clasa pentru simulare numita SimulationManager. In aceasta clasa am declarat toate atributele necesare procesului de distribuire al clientilor. Am implementat metoda de generare aleatoare pentru un numar de clienti dat de catre utilizator in interfata. Generarea randomizata a fost facuta in functie de parametrii introdusi de utilizator: timpul de sosire a fost generat aleator cu limite in timpul minim de sosire si cel maxim de sosire, timpul de procesare a fost generat aleator cu limitele in timpul minim de procesare si timpul maxim de procesare, iar id ul fiecarui client este reprezentat de un intreg care porneste de la 1 si care creste cu 1 pana la numarul maxim de clienti introdus de utilizator in interfata. Pentru a respecta timpul de sosire al clientilor am ordonat lista de clienti generati aleator crescator in functie de timpul de sosire. Pentru acest lucru am implementat un comparator direct in metoda de generare. Acest comparator consta in compararea timpului de sosire a doi clienti. Acest lucru se repeta pana cand lista va fi ordonata crescator in functie de timpul de sosire. In metoda de actiune ( run ) am pus in miscare simularea. Am asignat unei valori care reprezinta timpul curent valoarea 0. Cat timp timpul curent este mai mic decat timpul de simulare introdus de utilizator, in primul rand vom verifica daca exista un client cu timpul de sosire egal cu timpul curent, in caz ca exista va fi trimis pentru distriubuire cu ajutorul metodei dispatchTask care apeleaza la randul ei functia de addTask. Cresc timpul de asteptare si il impart la numarul de clienti care au fost distribuiti pana la momentul respectiv. Dupa toate acestea cresc timpul curent cu 1 si „opresc” ( sleep ) Thread – ul pentru o secunda ( 1000 ms ). Interfata am generat-o cu ajutorul metodelor implementate in swing. Pentru interfata destinata introducerii datelor am inserat 3 Combo Box ( se selecteaza valoarea ) si 5 campuri Text ( se scrie de la tastatura valoarea dorita ). Completarea respectiv selectarea unei valori in fiecare camp este necesara inaintea apasarii butonului Start. Acest buton deschide o fereastra noua numita Log View unde vor fi afisate plasarile in timp real a clientilor. Aceasta fereastra a fost creeata in clasa LogView. Aici am plasat 3 campuri de timp text, toate nemodificabile deoarece in aceasta fereastra sunt doar afisate plasarile. In campurile de sus se gases timpul mediu de asteptare si timpul curent. Acestea se actualizeaza in mod automat in fiecare secunda. Pentru unirea View- ului cu metodele implementate in Model am creat o clasa numita QueueManagementController in care am declarat un view nou. Am adaugat actiuni butonului de start astfel incat la apasarea butonului de Start se vor stoca toate valorile din campurile din interfata destinata introducerii datelor in niste variabile locale, variabile care se trimit ca parametrii pentru metoda de simulare, apoi creez un nou Thread asociat acelei functii si il pornesc.

5.Rezultate

Rezultatele programului sunt afisate in fereastra Log View. In cele doua campuri de sus apar timpul mediu de asteptare si timpul curent, iar in lista apar clientii introdusi la momentul respectiv. Rezultatele se actualizeaza in timp real, de la secunda la secunda. In lista clentii sunt introdusi cu setul de date : id, timp de sosire, timp de serviciu si coada in care au fost plasati ( ex: Clientul 2 4 10 a fost plasat in coada 2, adica clientul cu id 2, timp de sosire 4 si timp de procesare 10). Clientii nu vor aparea cu id urile in ordine deoarece acestia au fost generati aleator, fiind sortati ulterior dupa timpul de sosire.

6.Observatii

In urma acestui proiect am invatat sa lucrez cu Thread- uri, am invatat concepte noi cum ar fi Multi Threading, metode sincronizate. Am reactualizat cunostintele legate de Java: Interfete, ordonarea unei liste, implementarea unui log cu actualizare in timp real. Am inteles mai bine cum functioneaza listele, in special BlckingQueue.

7.Bibliografie

1. <https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_2.pdf>
2. <https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf>
3. <https://www.baeldung.com/java-synchronized-collections>
4. baeldung.com/java-synchronized
5. <https://www.baeldung.com/java-runnable-callable>
6. https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
7. <https://www.geeksforgeeks.org/>
8. https://stackoverflow.com/