Documentatie – Tema 2

Thread-uri

Nume: Dascal Raluca Georgiana

Grupa: 30226

Profesor coordonator: Dorin Moldovan

Cuprins:

1. Cerinte Functionale

Sa se realizeze o aplicatie care prin conectarea la o baza de date sa fie capabila sa adauge,,sa stearga si sa furnizeze anumite informatii despre elemtentele gasite in tabele acestei baze de date.

2. Obiective

2.1 Obiectiv Principal

Dezvoltarea unei aplicatii care sa stocheze intr-o baze de date informatii despre potentialii client, produsele din stoc, si comenzile pe care clientii le adauga. Aplicatia trebuie atat sa gestioneze tabelele din baza de date (adaugare,stergere actualizare a valorilor din tabele prezentate mai sus) cat si sa efectueze o factura (intru-un document pdf) sau sa afiseze un mesaj de eroare in cazul in care cantitatea ceruta pentru un produs este mai mare decat stocul disponibil. Pe langa aceasta functionalitate aplicatia trebuie sa mai respecte cateva reguli precum: folosirea programarii orientate pe obiect(concept aprofundat in semestrul trecut), crearea unor clase cu maxim 300 de linii de cod, implemantarea unor metode cu maxim 30 de linii de cod, folosirea conventiilor pentru nume Java, utilizarea unor fisiere text pentru citire si generarea unor documente PDF pentru returnarea detaliilor cerute, generarea fisierului JavaDoc precum si generarea unui fisier executabil .jar care sa respecte formatul de apel.

2.2 Obiective Secundare

Obiective secundare	Capitol
Descriera generala a aplicatiei	3
Descriera algoritmilor utilizati	5.2
Prezentarea claselor	6
Rezultatele obtinute	7

3. Descrierea Generala a Aplicatiei

Aceasta aplicatie este utila directorilor de centre comerciale deoarece permite stocarea informatiilor atat despre clientii lui, cat si despre produse si stoc. Aplicatia prezinta o baza de date dinamica, adica directorul poate cu usurinta adauga sau sterge un client sau un produs cand doreste. Actualizarea stocului se realizeaza automat in cazul in care directorul doreste sa introduca in stoc un produs deja existent, sau cand un client plaseaza o comanda. Mai mult de atat aceasta aplicatie genereaza la fiecare comanda plasaTA cate o facturA cu informatiile despre client,produs,cantitate si suma totala de plata. In cazul in care directorul doreste sa vada tabele, fara insa a accesa baza de date, el poate sa faca asta cu ajutorul unor intructiuni prezentate mai jos.

4. Analiza problemei

In figura de mai jos (fig 4.1) este descrisa intr-o forma grafica relatia dintre utilizator (directorii centrelor comerciale), aplicatie si furnizarea rezultatelor obtinute in urma parcurgerii unui algoritm. Am ales sa realizez aplicatie si in aceasta forma de prezentare pentru a fi mai usor de inteles pentru un potent utilizator atat din punct de vedere functional cat si din punct de vedere a datelor necesare pentru a putea fi utilizata.

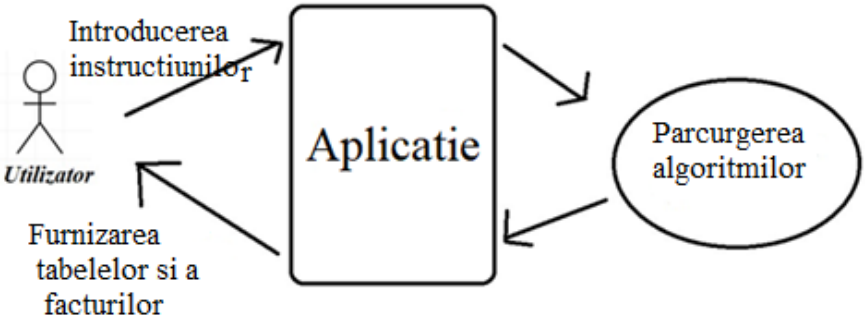


Fig 4.1

5. Proiectare

5.1 Diagrama de clase

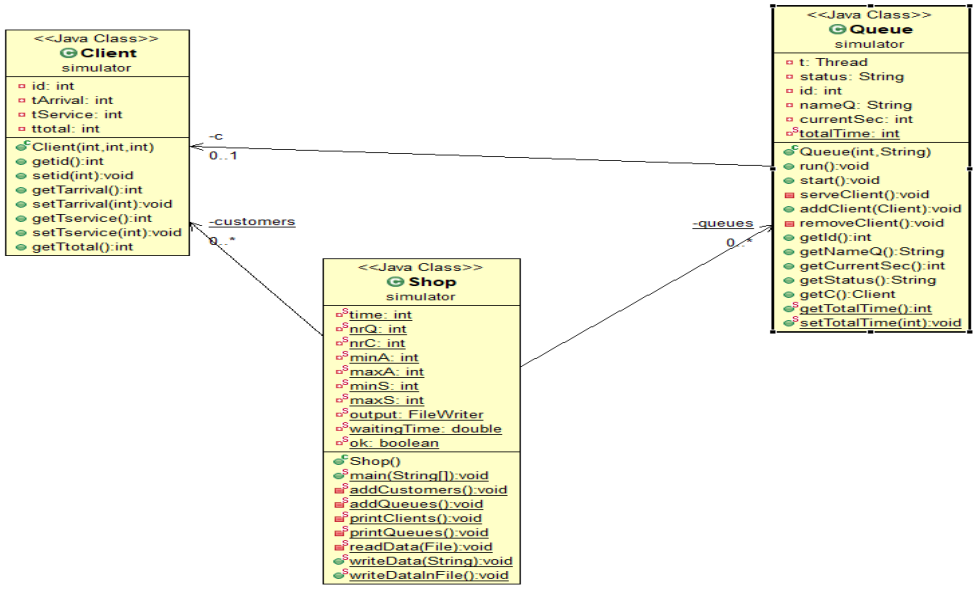


Fig 5.1.1

In figura de mai sus (fig 5.1.1) sunt prezentate diagramele UML a claselor proiectului realizat. Relatia dintre acestea a fost sintetizata de catre programul in care a fost scris si codul, Eclipse Java 2019. In aceasta imagine se evidentiaza atat fiecare clasa in parte (cu varabilele declarate si metodele ce definesc fie constructori, gettere ,settere fie algoritmi utili in realizarea functionalitatilor aplicatiei) cat si legaturile dintre acestea. Aceste diagrame UML ne ajuta la formarea unei imagini in ansamblu a proiectului ajutand totodata si la intelegerea unor concepte si paradigme a programarii orientate pe obiect.

5.2 Descrierea algoritmilor implementati

Fiecare dintre metodele definite si descrise in acesat aplicatie Java, cu exceptia constructorilor sau a celor care seteaza si returneaza valorile variabilelor declarate in clase, descriu cate o sarcina pe care proiectul trebuie sa o indeplineasca folosind un algoritm mai mult sau mai putin riguros.Algoritmii implementati in acest proiect au o saricna foarte bine definita si urmaresc realizarea functionalitatii corecte a cerintelor acestei teme. Descrierea implementarii si functionalizarii fiecare metode si utilitatea fiecarei variabile (aparute in diagram UML a claselor prezentata in capitolul anterior) vor fi prezentate in amanunt in urmatorul capitol.

6. Implementare

In acest capitol se va prezenta fiecare clasa a proiectului si se va explica utilitatea sau scopul fiecarei variabile declarate si metode implementate in aceste proiect avand la baza diagrama UML sintetizata de catre programul in care a fost scris codul. S-a pastrat un unic pachet (numit simulator) care contine toate cele trei clase necesare si create.

6.1. Clasa Client

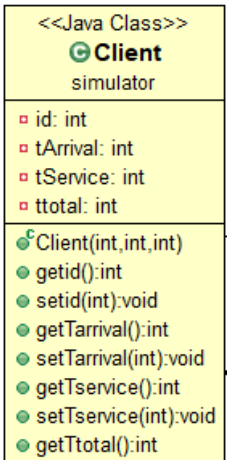
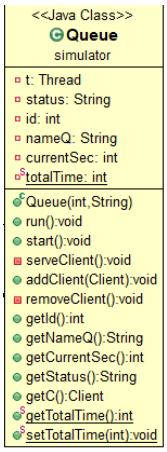


Fig 6.1

In figura alaturata (fig 6.1) este prezentata clasa Client care reprezinta clasa de temelie a proiectului. Aceasta clasa descrie personajul care ne intereseaza cel mai mult in acest proiect deoarece trebuie sa gasim o metoda eficienta prin care sa fim capabili sa il asezam la coada potrivita pentru ca acesta sa astepte cat mai putin timp. Clasa Client contine detalii despre acesta (Client) precum: un ID unic specific fiecarul client pentru a-l putea identifica cu usurinta, tArrival- timpul la care clientul si-a terminat activitatile si este gata sa se aseaza la una dintre cozi, tService- timpul pe care acesta il petrece in fata ghiseului/ casei de marcat (timpul de servire), ttotal-este o variabila definite ca finnd suma dintre cei doi timpi prezentati mai sus. Clasa contine metodele de baza des intalnite in limbajul de programare Java gettere si settere precum si un constructor explicit a acestei clase in care se calculeaza si timpul total pentru fiecare client conform regulilor prezentate mai sus.

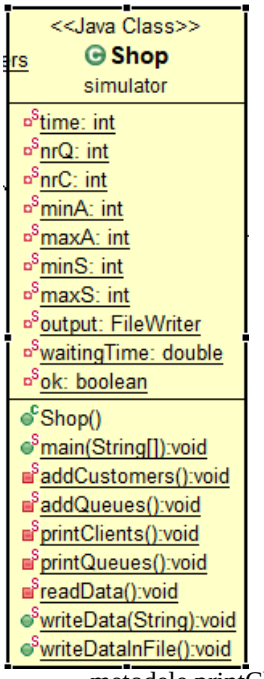
6.2. Clasa Queue



In figura 6.2 este explicata clasa Queue , clasa care cuprinde detaliile si functionalitatile fiecarei cozi. Coada conform cerintelor temei, trebuie sa fie reprezentata de catre un thread fiecare (declarat in prima varibaila t). Metodele run si start sunt specifice utilizarii thread-urilor si sunt implementate si in acesta aplicatie astfel incat sa se indeplineasca toate cerintele si functionalitatile intr-un mod corect. Variabila status se refera la starea actuala a unei cozi si poate avea una dintre urmatoarele valori closed si open, varibilele id si nameQ sunt parametrii unici de identificare a cozilor, variabila totalTime se refera la timpul de simulare a aplicatiei (citit ca paramtru din fisierul text de intrare) in timp ce variabila currentSec se refera la secunda in momentul rularii. Metodele serveClient, addClient, removeClient au un nume foarte sugestiv. Prima metoda presupune servirea unui client- transpus in termini mai utili implementarii-aceasta metoda se refera la actualizarea timpului de servire prin scaderea cu unu la fiecare secunda, in timp ce celalalte 2 metode presupun adaugarea si, respectiv stergerea clientului in respective din coada (in functie de momentele si valorile variabilelor necesare in luarea acestor decizii) si totoadata schimbarea starii cozilor din closed in open si invers. Celalalte metode sunt reprezintate de catre gettere si settere pentru anumite vairiabile. Metoda Queue reprezinta constructorul explicit al acestei clase.

Fig 6.2

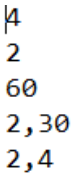
6.3. Clasa Shop



In figura alaturata (fig 6.3) este prezentata clasa principala a proiectului (cea care contine si metoda main), clasa Shop. In aceasta clasa se pun imbina toate rezultatele obtinute de catre celalalte clase si se realizeaza implementarea unei simulari de cozi, adica se pun bazele functionalitatii corecte a acestei teme. Explicarea variabilelor declarate in aceasta clasa; time- reprezinta secunda curenta a rularii, nrQ- numarul de cozi, nrC-numarul de clienti, minA, maxA- reprezinta timpul minim respective timpul maxim in care acestia se pot aseza la o coada, minS, maxS- reprezinta timpul minim si timpul maxim pe care un client il poate petrece in fata ghiseului/ casei de marcat. Toti acesti parametrii sunt cititi dintr-un fisier text si sunt furnizati algoritmilor pentru a-si putea indeplini saricinile. Variabila output de tip fileWriter reprezinta fisierul in care se vor scrie rezultatele proiectului. Variabila waitingTime reprezinta timpul mediu de asteptate a tuturor clientilor (acesta valoare se calculeaza ca fiind media aritmetica a tuturor timpilor de servire a clientilor si a timpilor de asteptare (daca este cazul) pentru fiecare client.. Si aceasta clasa contine un constructor explicit. Metoda readData citeste informatiile necesare din fisierul text in.txt si realizeaza furnizarea datelor catre alte metode (aceasta contine si distribuirea stringului si preluarea valorilor in variabilele corespunzatoare). Metoda addCustomers contine generarea random (aleatorie) a clientilor conform datelor cititie, adaugarea acestora intr-o lista si sortarea lor in ordine crescatoare in functie de timpul de ajungere. Metoda addQueues genereaza o lista cu elemete de tip coada. Metodele printClients si printQueues realizeaza afisarea detaliilor privind atat elementele din coada (in metoda prinQueues) cat si elementele despre clienti (in metoda printClients). Metoda writeData realizeaza afisarea datelor cerute de catre utilizator, furnizate de catre metodele printClients si printQueues conform formatului prezentat in cerintele temei. Metoda writeDataFile efectueaza afisarea datelor in fisierul text de iesire out.txt.

Fig 6.3

7.Rezultate



In figura alaturata(fig 7.1) este prezent fisierul in.txt realizat conform cerintelor. Datele din fisier reprezinta urmatoarele informatii (4- nr de client, 2- nr de cozi, 60- timpul de simulare, 2,30- timpul minim si timpul maxim de ajungere a clientilor la coada, 2-4- timpul minim si timpul maxim de asteptat pentru a realiza servirea fiecarui client).

Fig 7.1

In figura de mai jos (7.2) este prezentat fisierul out.txt, fisierul de scriere a aplicatiei. In urmatoare imagine se poate verifica atat corectitudinea datelor furnizate spre afisare cat si respectarea formatului cerut in aceasta tema. Afisarea se opreste in in urmatoarele doua conditii: - toate cozile sunt inchise (adica au stasul closed) si nu exista alti client in lista, sau – cand timpul curent al rularii este mai mare decat timpul de simulare. Ultima linie din fisier reprezinta timpul mediu pe care clientii l-au asteptat fie la coada, fie in fata ghiseului/ casei de marcat. Rezultat obtinut este corect conform regulilor prezentate mai sus.

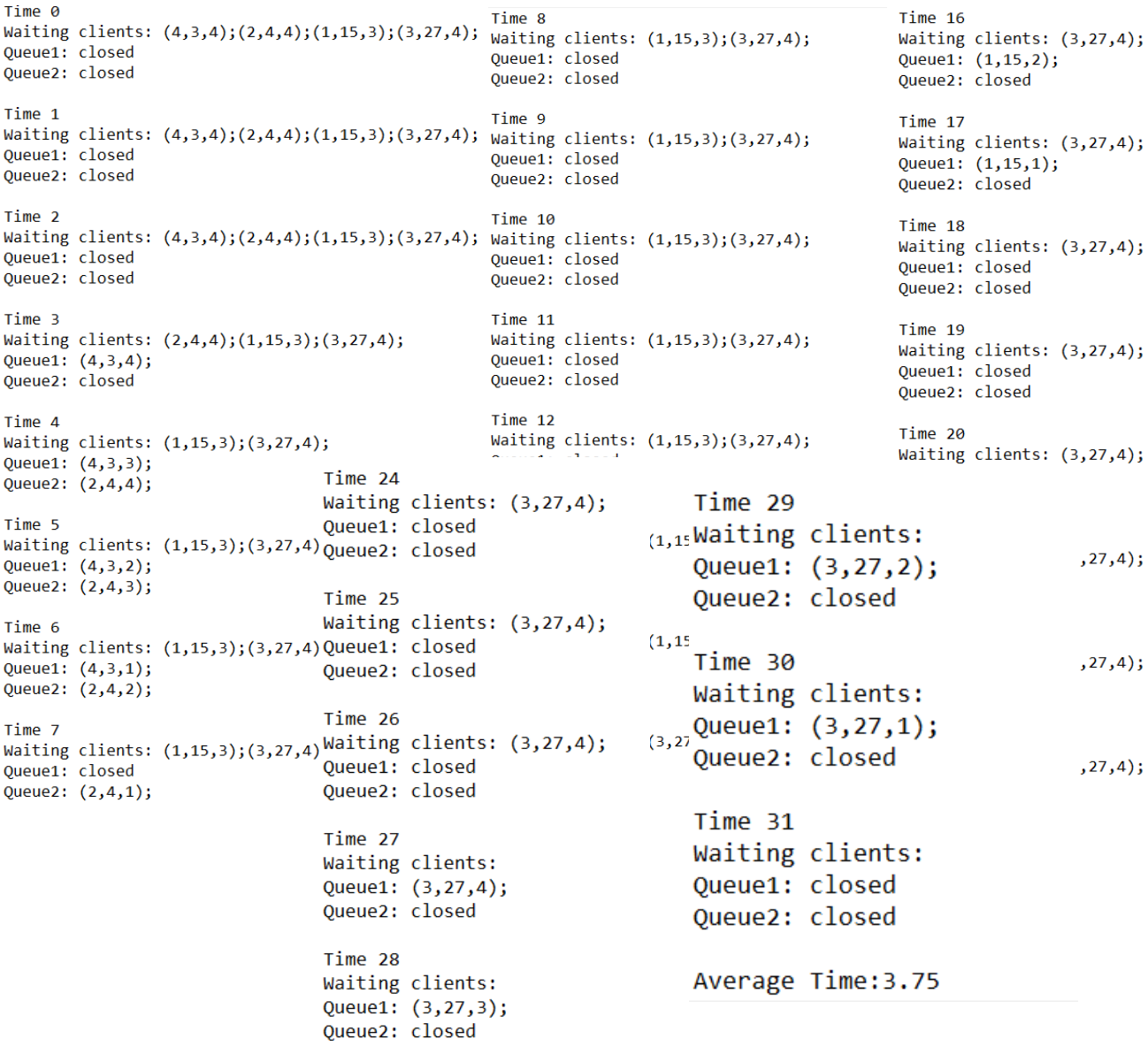

Time 0
Waiting clients: (4,3,4);(2,4,4);(1,15,3);(3,27,4);
Queue1: closed
Queue2: closed
Time 1
Waiting clients: (4,3,4);(2,4,4);(1,15,3);(3,27,4);
Queue1: closed
Queue2: closed
Time 2
Waiting clients: (4,3,4);(2,4,4);(1,15,3);(3,27,4);
Queue1: closed
Queue2: closed
Time 3
Waiting clients: (2,4,4);(1,15,3);(3,27,4);
Queue1: (4,3,4);
Queue2: closed
Time 4
Waiting clients: (1,15,3);(3,27,4);
Queue1: (4,3,3);
Queue2: (2,4,4);
Time 5
Waiting clients: (1,15,3);(3,27,4)
Queue1: (4,3,2);
Queue2: (2,4,3);
Time 6
Waiting clients: (1,15,3);(3,27,4)
Queue1: (4,3,1);
Queue2: (2,4,2);
Time 7
Waiting clients: (1,15,3);(3,27,4)
Queue1: closed
Queue2: (2,4,1);
Time 8
Waiting clients: (1,15,3);(3,27,4);
Queue1: closed
Queue2: closed
Time 9
Waiting clients: (1,15,3);(3,27,4);
Queue1: closed
Queue2: closed
Time 10
Waiting clients: (1,15,3);(3,27,4);
Queue1: closed
Queue2: closed
Time 11
Waiting clients: (1,15,3);(3,27,4);
Queue1: closed
Queue2: closed
Time 12
Waiting clients: (1,15,3);(3,27,4);
Time 16
Waiting clients: (3,27,4);
Queue1: (1,15,2);
Queue2: closed
Time 17
Waiting clients: (3,27,4);
Queue1: (1,15,1);
Queue2: closed
Time 18
Waiting clients: (3,27,4);
Queue1: closed
Queue2: closed
Time 19
Waiting clients: (3,27,4);
Queue1: closed
Queue2: closed
Time 20
Waiting clients: (3,27,4);
Time 24
Waiting clients: (3,27,4);
Queue1: closed
Queue2: closed
Time 25
Waiting clients: (3,27,4);
Queue1: closed
Queue2: closed
Time 26
Waiting clients: (3,27,4);
Queue1: closed
Queue2: closed
Time 27
Waiting clients:
Queue1: (3,27,4);
Queue2: closed
Time 28
Waiting clients:
Queue1: (3,27,3);
Queue2: closed
Time 29
Waiting clients:
Queue1: (3,27,2);
Queue2: closed
Time 30
Waiting clients:
Queue1: (3,27,1);
Queue2: closed
Time 31
Waiting clients:
Queue1: closed
Queue2: closed
Average Time:3.75

Fig 7.2

8.Concluzii si posibilitati de dezvoltare ulterioare

In timpul dezvoltarii acestei teme, adica in timpul realizarii unei aplicatii care simuleaza repartizarea unor numar exact de clienti la un anumit numar de cozi astfel incat timpul de asteptare a acestora sa fie minim, consider ca am reusit sa aprofundez atat unele paradigme si concepte ale programarii orientate pe obiect cat si imbunatariea abilitatile mele de a scrie cod si/ sau a corecta eventualele erori atat de sintaxa cat si la nivel de gandire aparute pe parcursul descrierii algoritmilor necesare functionalitatilor acestei aplicatii. Pe langa acestea am realizat cat de importanta este citirea cu atentie si intelegerea unei cerinte si capabilitatea de a structura/ imparti problema initiala in mai multe probleme mai mici si inceperea rezolvarii acestora ca un prim pas in indeplinirea sarcinilor unei teme,ca apoi sa asociezi toate rezultatele obtinute din surse pentru a ajunge la un rezultat corect in ansamblu. Am aprofundat si cunostiintele mele atat teoretice cat si practice in ceea ce privesc thread-urile si efectuarea unor operatii cu acestea. Mi-am imbunatatit si abilitatea de a cauta informatii utile si corecte pentru a reusi ducerea proiectului meu intr-o stare mai complexa si mai corecta din punct de vedere a functionalitatii.

O posibilitate de dezvoltare a proiect realizat si descris in acesta documentatie ar putea fi:

- Implementarea unui algoritm care nu necesita atat de multe informatii (precum cei 2 parametrii pentru client- timpul de ajungere si timpul de servire a acetuia) ca aceasta sa poate fi folosita chiar si de catre detinatorii unor spatii comerciale sau institutii care lucreaza cu clientii si doresc o imbunatatire a eficientei serviciilor si o multumire superioara adusa clientilor.
- mai mult de atat s-ar putea face o revizuire asupra tuturor algoritmilor folositi si realizarea implementarii acestora intr-o maniera mai eficienta din punct de vedere a timpului de executie sau a memoriei folosite sau mai usor de descris (pentru programator) folosind functii deja definite ale pachetelor Java, astfel reusind o reducere a timpului petrecut pentru asigurarea functionarii unui astfel de aplicatii.
- o alta posibilitate de dezvoltare a proiectului ar putea fi imbunatatirea proiectului realizand chiar si o interfata grafica pentru ca utilizatorului sa-I fie cat mai usor intelegerea rezultatelor si a utilizarii aplicatiei

8. Bibliografie

https://www.tutorialspoint.com/java/java_multithreading.htm

https://conspecte.com/Programare-Java/java-citirea-datelor-de-la-tastatura-si-afisarea-lor.html

https://www.w3schools.com/java/java_arrays.asp