**Documentatie-Tema 2**

Hitu Octavian

Grupa : 30226

Profesor Laborator : Moldovan Dorin Vasile

**Cuprins**

1. Obiectivul Temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
   1. Analiza Problemei
   2. Modelare
   3. Cazuri de utilizare
3. Proiectare
4. Diagrama UML
5. Clasa Client
6. Clasa Coada
7. Clasa ManagerCoada
8. Clasa View
9. Clasa Controller
10. Rezultate
11. Concluzii

1.Obiectivul temei

Proiectul pe care l-am implementat reda intr-o forma simpla , un sistem dianmic , acest tip de system regasindu-se peste tot in lume. In proiect putem vedea o simulare de a cozilor, in care clientii sunt generate , dupa care asezati la cozi , dupa un algoritm care determina cea mai mica coada dintre un sir de cozi. In capitolele ce urmeaza, voi exeplica cum se poate simulare o coada ,cum s-a calculate timpul mediu de asteptare si cum am adaugat cate un client nou in cea mai scurta coada. Pentru a intelege toate aceste aspect , noi trebuie sa intelegem mai intai ce inseamna conceptul de multithreading, acesta fiind folosit in program.

Java este un limbaj de programare multi-threaded, ceea ce înseamnă că putem dezvolta un program multi-threaded folosind Java. Un program multi-threaded contine două sau mai multe componente care pot rula simultan si fiecare parte poate sa se ocupe de o alta sarcina, facand în acelasi timp o utilizare optima a resurselor disponibile, mai ales atunci când calculatorul are mai multe CPU-uri.

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

2.1. Analiza Problemei

Daca este sa ne gandim intr-o situatie reala, problema proiectului nostru se regaseste peste tot in viata noastra, de exemplu in magazine sau in orice institutie in care avem foarte multi clienti si un numar limitat de ghisee/case ,de aici reiesind problema cozilor simultane. Servirea primilor clienti din fiecare coada se face in acelasi timp de aici se pune problema efectuarii anumitor task-uri in paralel.

In limbajul de programare Java, pentru a exprima cat mai concret o asemenea situatie, avem posibilitatea de a folosi Threads, lucru pe care l-am detaliat în capitolul anterior.

2.2. Modelare

Modelarea aplicatiei se bazeaza pe conceptele programarii orientate pe obiecte, acestea fiind:

Clasele- prin clasa intelegem o forma a unui obiect deoarece aceasta reprezinta o descriere generalizata a acestuia. Clasa defineste toate atributele pe care un obiect le are ,iar metodele ce fac parte din acea clasa definesc functionalitatea acelui obiect;

Obiect- un obiect reprezinta o componenta software care incorporeaza atribute si operatii . Operatiile se pot efectua asupra atributelor .

Incapsularea- aceasta are rolul de a uni informatiile importante ale unui obiect. Ea totodata restrictioneza accesul la date si metode din exterior.

Mostenirea- din punct de vedere ierarhic aceasta permite claselor similare sa se suprapuna. Datorita ei clasele inferioare pot folosi metode din clasele superioare.

Polimorfismul- prin definitie aceasta reprezinta abilitate de a lua mai multe forme .Prin polimorfism aceeasi metoda ce este folosita intr-o superclasa poate fi suprascrisa fiind reutilizara in subclase ,ele avand o functionalitate diferita

2.3. Cazuri de utilizare

Aplicatia dispune de o interfata care permite introducerea unor parametri de intrare de la tastatura, aceasta ducand la o varietate larga de situatii de simulare.

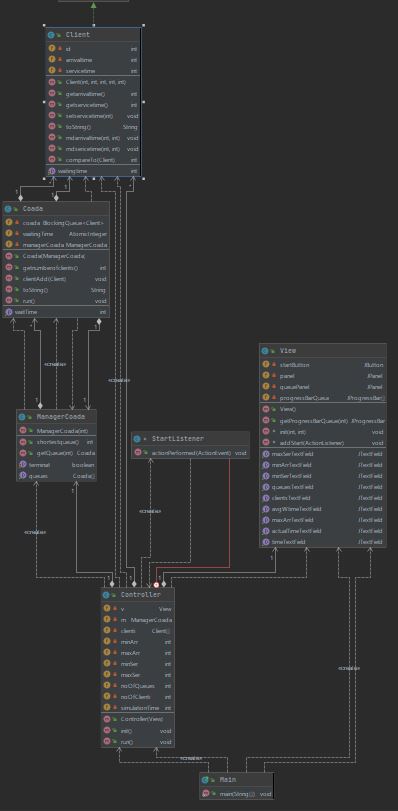
In randurile urmatoare voi prezenata parametrii de intrare din interfata grafica a aplicatiei:

* Primii doi parametri care vor fi introdusi sunt „Min Arrival Time” si „Max Arrival Time”. Acestia permit setarea unui interval de timp in interiorul caruia se va genera o valoare aleatoarie de timp ce se va asigna fiecarui client ,acesta reprezentand momentul aparitiei lui in aplicatie.
* Urmatorii doi parametri care vor fi introdusi sunt „Min Serice Time” si „Max Serice Time”. Acestia permit setarea unui interval de timp in interiorulc aruia se va genera o valoare aleatorie ce se va asigna fiecarui client aceasta reprezentand timpul de asteptare al acestuia in fata cozii , pana la iesirea din ea.
* Un alt parametru este ”Time to Simulare” care reprezinta timpul total de simulare. In momentul in care numarul de secunde vor egala timpul total de simulare, simularea se va incheia.
* Urmatorul parametru il reprezinta „Number of Queues”. Acesta este numarul de cozi care vor fi generate in simulare.
* Ultimul paarmetru este „Number of Clients”.Acesta va fi numarul de clineti care vor fi generati in simulare.

3.Proiectare

Aplicatia are la baza 6 clase dintre care una este cea de main. Acesta este organizata confrom sablonului Model-View-Controller (MVC). Acest tip de organizare reprezinta separarea interfetei utilizator într-o vedere (View) care interactioneaza cu Modelul după nevoi, si un Controller care raspunde la cererile utilizatorului, interactionand atat cu View-ul, cat si cu Modelul.

3.1. Diagrama UML



3.2. Clasa Client

Aceasta clasa contine atributele destinate fiecarui client, acestea fiind id-ul clientului pentru a putea fi intendificat ,timpul in care acesta a fost geneat, timpul de service si timpul de asteptare al acestuia al acestuia. Aici au fost implementate metode de get si set ,dar si metoda de toString care scrie id-il , arrival time-ul si service time-ul. In aceasta clasa se gasesc si cele doua metode care genereaza random arrival time-ul si service time-ul pentru fiecare client in parte dupa datele de timp introduse de utilizator in interfata.

3.3. Clasa Coada

In cadrul acestei clase care extinde Thread , s-a creat o colectie de tipul Blockingqueues de tip client ,un AtomicInteger care ne va ajuta sa calculam waiting time-ul si o variabila de tip manaegrcoada. Pe langa metodele de get mai avem si metoda clientadd , care asa cum sugereaza numele adauga clientii in coada, setand waiting time-ul ,dupa care adaugand-ul si la finalul metodei se adauga clientul in coada. Mai avem metoda toString care ne va afisa id-ul clientului, arrival time-ul si service time-ul acestuia folosind metoda cu acelasi unme din clasa client. Totodata avem implemenatat si Run-ul in interiorul caruia se declara un client nou si atat timp cat timpul de simulare nu este gata se vor face urmataorele actiuni; dupa ce s-a verificat daca coada nu este goala intr-un bloc try-catch se va lua primul client din coada si intr-o instructiune while sevice time-ul se va decrementa pana cand ajunge la 0; aici se va termina while-ul ,iar clientul nostru, al carui service time a exprirat va fi scos din coada.

3.4. Clasa ManagerCoada

In cadrul acestei clase au fost declare un vector de tip coada si o valoare booleana care va determina cand timpul de simulare este terminat. O prima metoda pe care o gasim in clasa este cea care prin intermediul numarului de cozi creeaza cate o coada pentru fiecare element din vectorul de tip Coada declarat anterior. Mai putem gasi functia „”shortestqueue” care ,dupa cum sugereaza numele, prin tntermediul unei instructuni for va returna numarul celei mai scurte cozi din aplicatie, pentru a putea adauga ulterior urmmatorul client . Aceste metode sunt urmate de alte get-ere si set-ere.

3.5 Clasa VIew

Aceasta este una din clasele care care fac posibila implemenatrea aplicatiei dupa sablonul Model-View-Controlller. Intial aici au fost declarate text Panel-urile, Text field-urile si buton care vor apare in interfata. Aici se regaseste si un sir de progress abr-uri care vor reprenta cozile, fiecare dansusi update in timp real. Pe langa get-ere si set-ere avem metoda de view in care s-a creat afisarea propriuzisa a tuturor elemntelor precizate anterior, ele incadrandu-se in panel-ul mare. Totodata avem si init-ul caruia ii sunt date ca parametri numarul de cozi si numarul de clienti. Intr-o instructiune de tip for care va avea ca numar de iteratii numarul de cozi , va crea pentru fiecare coada un panel in care va fi nuamrul cozii si progress bar-ul ei care se va updat-a in timp real,valorile din ea fiind cuprinse intre 0 si numarul de clienti.

3.6. Clasa Controller

Clasa Controller la fel ca si cea VIew face posinila implementarea aplicatiei dupa sablonul Model-View-Controller. Aici am declarat intial VIew-ul, o varaibila de tip ManagerCoada si o lista de clienti, dupa care am decalarat variabilele care vor urma sa fie citite de la tastatura in interfata . Acestea sunt minArr si maxArr care vor reprezenta minimum arrival time si maximum arrival time , minSer si maxSer ,care reprezinta minimum service time si maximum service time, noOfQueues (nuamrul de cozi) , noOfClients (numarul de clienti) si simulationTime (timpul in care va avea loc simualarea). In interior acestei clase putem gasi initial metoda init in care am declarat o variabila managercoada care va contiene un sir de cozi si o variabila client care av contine un sir de clienti. Vom parcurge cu o instructiune for de la 0 pana la numarul de clienti si vom iniatiza fiecare client cu id-ul sau si i se va asigna un arrival time si un service time calculat cu moteode din clasa client. Tot aici ii vom sorta si vom initializa view-ul cu nuamarul de cozi si numarul de clienti , la final pornind thread-ul. In aceasta clasa avem totodata metoda run in care am implemenatat un bloc try-catch. In acest bloc vom face scrierea in fisier care va fi gasit in folder-ul in care se afla proiectul si vomimplementa rularea aplicatiei. Verificand daca timpul propus pentru simulare este gata vom parcurge clientii si verificand daca arrival time-ul este egal cu timpul actual al simularii ii vom pune pe rand in cea mai scurta coada la acel timp de simulare, incepand totodata scriereain fisier. Tot in instructiunea while vom face o alta instructiune for de la 0 la nuamrul de clienti si verificam daca arrival time-ul este mia mare decat timpula actual pentru a-i putea scrie in fisier. Dupa ce au fost scrisi clientii in asteptare vom face doua for-uri, unul dintre ele va scrie cozile in fisier ,iar celalt se va actualiza progress bar-ul. La finalul while-ului se ca actualiza valoarea timpului de simulare actual si se va incremenata. While-ul s-a terminat si revenim in try-catch unde se va calcula average waiting time-ul prin adunarea tuturor waiting time-urilor clientilor si imaprtirea la nuamrul de clienti. La final aceasta valoare se va afisa ,fisierul se va inchide si se va seta timpul de executie al simualrii ca terminat. Putem observa ca in aceast clasamai avem o subclasa ce implementeaza un listener ,acesta luand valorile scrise de catre utilizator de la tastatura si atribuindu-le variabilelor decarate mia sus in aceasta clasa.

4. Rezultate

Rezultatele acestui proiect pot fi vazute doar prin pornirea aplicatiei, pentru ca doar astfel putem vedea cum se aseaza clientii la coada , asteptand sa ajunga la capatul cozii ,unde va astepta un timp de servire aleatoriu ,apoi va fi scos din coada. Aplicatia va genera un numar de clienti pe care utilizatorul il va da de la tastatura , si va genera numarul de cozi pe care utilizatorul va dori sa le aiba simulare. Dupa ce toti clientii ai fost generati acesti vor intra in cozi in functie de timpul de sosire si de coada care are cei mai putini clienti la coada. Aplicatia ruleaz bine ,explficand prin fisierul text pasii de la fiecare timp de executie . Dupa crearea fiecarei clase descrise in capitolele de mai sus si rularea aplicatiei , pot spune ca proiectul simuleaza foarte bine un sistem bazat pe cozi. Acest proiect are o interfata usor de utilizat, avand numai un buton de start si field-urile in care trebuie sa introducem valorile destinate simularii.

5. Concluzii

Prin acest proiect , am putut sa vad concret si sa inteleg avantajul abstractizarii inca din primele faze ale dezvoltarii acestei aplicatii. Avand o dificultate mai ridicata decat tema precedenta ,pot spune ca am invatat importanta organizarii timpului acordat unui proiect, structurarea sa si analizarea cu atentie a problemei inainte de inceperea implementarii propriuzisa a proiectului.

Din punctul meu de vedere, posibilele dezvoltari care pot fi aduse aplicatiei ar fi : imbunatatirea interfetei din punct de vedere estetic pentru a fi mai usor de inteles de catre utilizator, folosind anumite imagini sau forme in locul acelor progress bar-uri facute pentru reprezentarea cozilor, stabilizarea unor exceptii mai drastice in ceea ce priveste introducerea valorilor de la tastatura sau afisarea unor erori in cazul introducerii altor simboluri de la tastatura, astfel prevenind neatentia utilizatorului.

In concluzie, cred ca am respectat conditiile prestabilite pentru aceasta aplicatie si ca acest proiect simuleaza o situatie reala intr-un mod concret fara alte probleme. Desi afisarea este simplista ,ea descrie corect si inteligibill o situatie reala.