

DOCUMENTATIE

TEMA 2

Luca Andrada Aurelia

Grupa 30228

CUPRINS

**1.** Obiectivul temei

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

**3.** Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

**4.** Implementare si testare

**5.** Rezultate

**6.** Concluzii, ce s-a invatat din tema, dezvoltari ulterioare

**7.** Bibliografie

**OBIECTIVUL TEMEI**

**Cerinta:**

Tema 2, cu titlul „Analiza unui sistem de cozi” are ca obiectiv proiectarea si implementarea unei aplicatii cu ajutorul careia sa se faca o simulare a unui sistem de cozi.

**Obiective:**

Realizarea unei aplicatii Java care se realizeze o simulare a unui sitem de cozi

* Citirea datelor despre clienti.
* Repartitia clientiilor astfel incat fiecare client sa fie asezat la coada cea mai optima din punct de vedere al timpului de servire
* Calcularea timpului mediu de asteptare pentru fiecare client si orele de varf
* Implementarea unui interfete grafice cu utilizatorul
* Testarea aplicatie Java, folosind JUnit.

**2. Analiza problemei si modelarea acesteia**

Prin analiza problemei, ne referim la un prim set abstract de operatii si proprietati prin care încercam sa depistam eventualele însusiri și comportamente ale proceselor necunoscute. Programarea orientata ne oferă aici un avantaj clar, tocmai fiindcă ea permite să taclam problema de la un nivel superiror, fara a mai fi constrasi, într-o asa masura, de caracteristicile tehnice.

Această strategie de conceptualizare, mai poartă numele si de bottom-up design. Este foarte avantajoasa din prisma găsirii componentelor constituente, deoarece pot fi gasite, relativ usor, structuri cu o legatura directă în lumea reala( obiecte, acțiuni etc.).

De obicei se porneste da le documentatia proiectului:

* Cautam substantive legate de proiect (in cazul nostru Client ,Caoda si Simulator)
* Cautam verbe legate de proiect (in cazul nostru: adauga client , creeaza coada, sterge Client, etc)

De ce cautam substantive si verbe? Deoarece substantivele sunt candidatii perfecti pentru a deveni clase, iar verbele pentru a deveni metode.

Odata realizat pasul de mai sus, ar trebui să avem o idee foarte generala asupra problemei. Pasul urmator constă în descrierea functionala a acesteia

Conceptul de coada este adesea intalnit si in viata reala. Astfel, va fi utilizat de multe ori si in cadrul aplicatiilor realizate in limbaje de programare, care simuleaza un comportament din realitate. Cozile sunt structuri ce contin date de acelasi tip. Ca operatii de baza asupra lor, avem adaugarea de elemente si stergerea acestora. Cozile functioneaza dupa principiul First In First Out, astfel incat al n-lea element introdus in coada va fi al n-lea care iese.

In aplicatia noastra, cozile trebuie sa reprezinte locuri unde clientii vor putea sa astepte, cu scopul de a primi un anumit serviciu. Un bun exemplu din realitate ar fi casele din magazinele de tip supermaket. Scopul principal al proiectului este minimizarea timpului de asteptare pentru fiecare client in parte. Totusi, se doreste ca nici costul sistemului sa nu fie prea mare , in cazul in care nu este necesar. Astfel, va trebui sa gasim o cale optima intre a servi clientii intr-un timp cat mai scurt, si a pastra costul relativ mic ( costul creste prin deschiderea de noi cozi ). In fiecare moment al rularii, clientii se vor imparti uniform intre cozile deschise pentru ca timpul lor de asteptare sa fie minim, ceea ce inseamna ca toate cozile disponibile ar trebui sa fie ocupate, daca numarul clientilor este mai mare decat numarul lor.

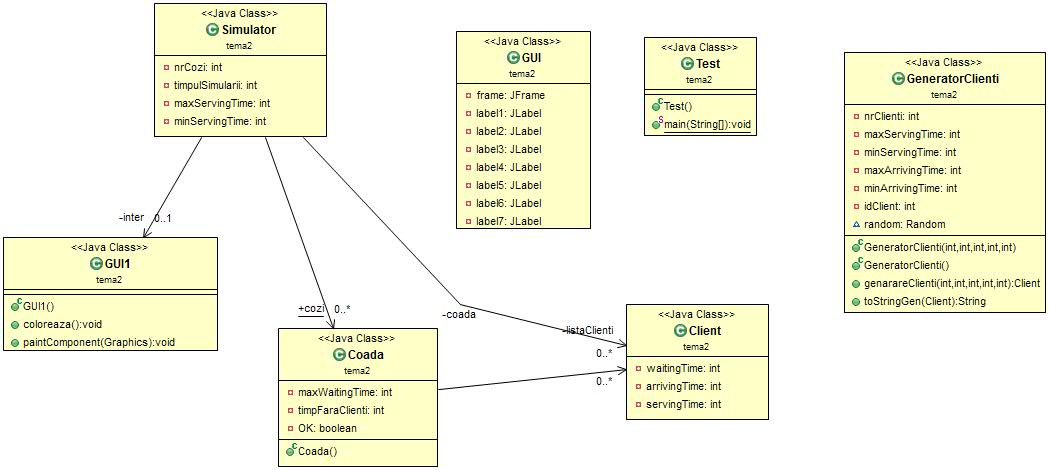
Acest proiect va simula o serie de clienti ce vor ajunge la cozi intr-un anumit interval de timp. Dupa sosire, ei se vor aseza la coada unde au cel mai putin de asteptat, vor astepta sa le vina randul iar dupa servire vor iesi din coada. Pentru a realiza aceste lucruri, trebuie sa cunoastem momentul in care fiecare client soseste la casa si timpul de servire de care are nevoie. Despre cozi va trebui sa stim la fiecare dintre ele cat timp trebuie sa astepte noul client sosit pana cand va ajunge in varful ei, unde i se va oferi serviciul, deoarece acesta doreste sa astepte cat mai putin. Acest timp depinde de numarul de clienti de la fiecare coada si de timpul de care are nevoie fiecare in momentul in care ajunge in capul cozii.

Ca si date de intrare, avem intervalul in care vor sosi clientii, si anume limita minima si maxim a acestuia. Pe langa asta, mai trebuie sa stim intervalul in care se va incadra timpul de servire pentru fiecare client, numarul de clienti ce vor sosi la cozi in total, numarul total de cozi disponibile precum si perioada de timp pe parcursul careia se va face rularea.

La iesire, vom afisa in primul rand cozile si clientii care se afla la acestea, in timp real, iar la finalul rularii aplicatiei se vor afisa timpul mediu de servire si ora de varf.

Pentru o usoara utilizare a aplicatiei, am ales sa implementam o interfata grafica relativ usor de folosit. Astfel, vom avea locatii in care introducem datele de intrare, vom avea un buton cu ajutorul caruia pornim simularea. Mai mult, vom avea campuri separate pentru fiecare informatie in parte. Asta inseamna ca vom putea urmari in timp real clientii care urmeaza sa intre in cozi, clientii care au fost repartizati deja si asteapta, precum si cei care au primit deja seriviciul si au parasit cozile de asteptare. In realizarea acestei interfete s-au folosit componente precum Jframe, Jlabel-uri, butoane sau JtextArea.

3. PROIECTARE



**Structuri de date:**

Cum a fost precizat si anterior, modelul de implementare a aplicatiei este inspirat din viata reala . Clientul este unitatea de baza a aplicatiei . Acesta contine doar informatiile necesare identificarii si gestionarii lui . Cozile implementate pentru a functiona depinzand de magazin , dar independent una de celelalte ( fiecare coada are firul propriu de executie ) . Structura de date folosita pentru stocarea clientilor intr-o coada este „ LinkedList ” . Am ales aceasta structura deoarece putem folosi metodele **add(<T> x)** ( adauga un element de tipul <T> la coada ), **removeFirst()** ( elimina si returneaza primul element din coada ) si **get(i)** ( returneaza elementul de la indexul i ) , metode care ne ajuta la implementarea cozii .

Intr-un magazin sunt mai multe cozi , stocate intr-un tablou simplu . Acest tablou a fost ales datorita modului usor si direct de a accesa informatiile .

Numarul de clienti este necesar pentru crearea unei simulari , dar cum aceasta data nu este introdusa de utilizator , in functie de durata simularii am creat un „ pool ” de clienti la inceputul simularii , acest pool fiind stocat intr-un arrayList.

**Proiectarea claserlor**

Punctul de pornire este reprezentat de clasa Client. Aceasta modeleaza un client din viata reala avand atributele necesare , printre care se numara timpul de sosire, timpul necesar servirii sau ID-ul pentru identificare. Metodele acestei clase sunt de tip get si set, ce ne vor ajuta sa citim sau sa modificam atributele.

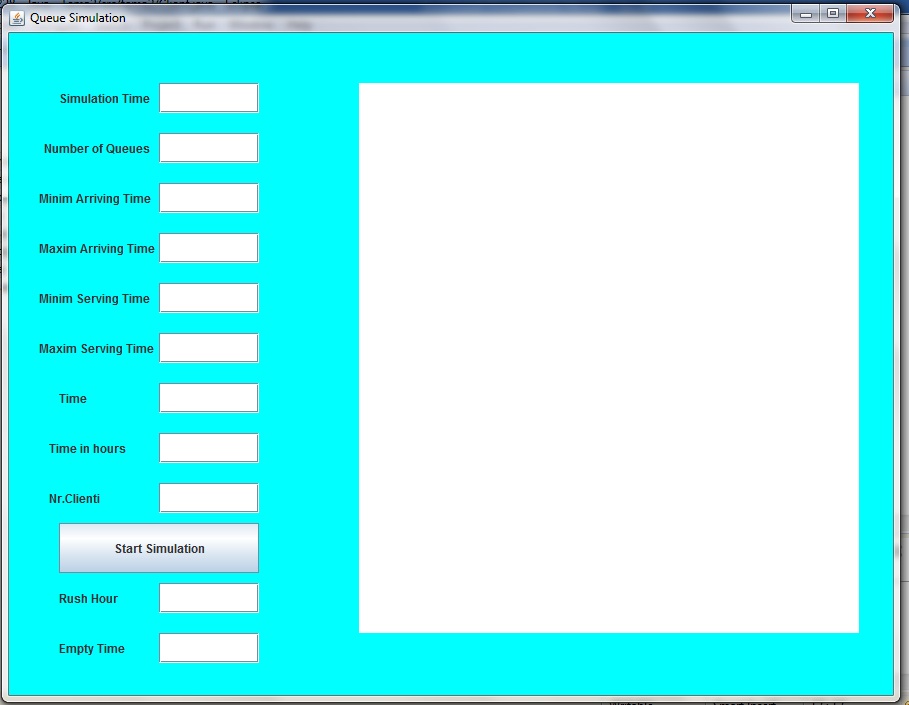
In continuare, descoperim clasa GeneratorClienti, care ne va genera clienti (care vor fi adaugati intr-o lista de clienti in clasa Simulator) in functie de parametri precum timpul minim si maxim de servire sau intervalul de sosire al acestora.

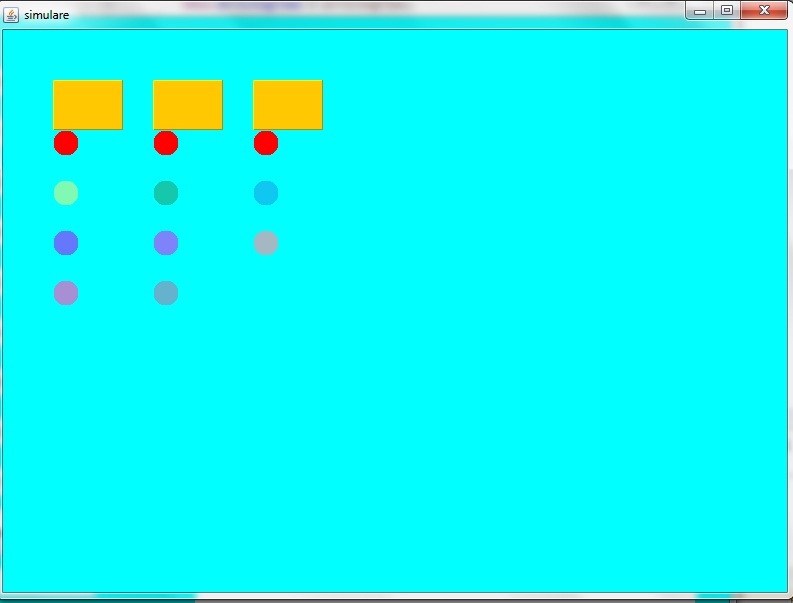
Mai departe, avem clasa Coada, care iarasi modeleaza obiecte din viata reala. Ca si atribute, fiecare coada este alcatuita dintr-o lista de clienti, si ii putem atribui un timp total de asteptare care este calculat prin insumarea timpilor de servire pentru fiecare client in parte. Metodele acestei clase sunt de tip get si set, pentru a citi sau modifica atributele.

Acum ca am prezentat clasele de mai sus, trecem la una care se foloseste de ele, si anume Simulator. In aceasta este descrisa o parte din functionaliatea proiectului. Ea contine obiecte de tip GeneratorClienti, lista de clienti sau cozi, iar metodele ne ajuta sa gasim clientii sositi sau serviti deja, sa-i afisam pe cei de la o anumita coada sau sa calculam unde va fi inserat urmatorul clienti, precum sa-i si inseram sau stergem. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**Interfata grafica**

Dupa cum se poate observa iterfata grafica este foarte simpla si usor de inteles. In partea stanga avem TextField-uri in care v-om introduce datele de intrare(timpul simularii, numarul de cozi, atat timpul minim si timpul maxim in care vor ajunge clientii cat si timpul minim si timpul maxim de servire al acestora dar si numarul de clienti care vor fi serviti. Avem un buton pentru inceperea simularii si cateva casute i care ni se arata timpul curenta al simularii, la terminarea ei ora de varf si timpul in care casele nu au avut nici un client (Empty Time). Iar in partea dreapta avem un AreaText in care ni se vor arata pe tot parcursul simularii informatii despre clientii: timpul de sosire , timpul de servire , la ce coada au fost asezati dar si timpul de plecare de la coada.





Fiecare patrat/dreptunghi reprezinta o casa. Acestea la inceputul simularii sunt toate gri cea ce inseamna ca nu sunt deschise / nu au clienti. In momentul in care un client se aseaza la coada aceasta automat se face galbena. Primi clienti de la coada tot timpul vor fi rosii ceilalti primesc culori generate rondom.

**Algoritmi**

Scopul mini-proiectului a fost de a invata modul de implementare si folosire a firelor de executie . Majoritatea codului scris a fost de fapt doar preluarea datelor de la utilizator si afisarea simularii in diferite formate ( grafic si text ) .

Algoritmii au fost necesari pentru a gestiona clientii in cadrul cozilor ( asteptarea pentru procesarea produselor fiecarui client ) si pentru alegerea cozii optime la care sa se aseze clientul .

Desi asezarea la coada nu este o alegere foarte importanta in viata reala , alegerea unei cozi optime duce la terminarea cumparaturilor mai devreme sau mai tarziu . Aceasta modalitate de a alege coada optima poate fi implementata in diferite moduri , dar modelul ales a fost ales pe baza dimensiunii cozii . Cand un client ajunge la coada , acesta isi alege coada cea mai scurta si se aseaza la aceasta .

**Implementare**

Mediul de dezvoltare ales pentru implementare este Eclipse IDE datorita multor facilitati pe care acesta le pune la dispoizita programatorilor. Cateva din acestea sunt : interfata intuitiva cu utilizatorul, grad mare de customizare, functii pentru scrierea mai usoara a codului (auto-complete, auto-import, auto-indent), crearea usoara si rapida a interfetelor grafice cu ajutorul windowbuilder-ului.

Modul in care a fost implementata aplicatia este urmatorul : utilizatorul introduce date despre simularea pe care vrea sa o creeze si apasa butonul „ Start Simulation ” . Aplicatia produce imediat o simulare care sa corespunda cu datele introduse de utilizator .

Cat timp cozile sunt goale , acesta asteapta sa primeasca clienti . Acest timp de asteptare este contorizat deoarece este necesat ca la finalul simularii sa afisam timpul cat au stat cozile goale . Am ales sa afisez timpul mediu in care cozile au fost goale ( timpul total / numar cozi ) . Cat timp au clienti, cozile

Fiecare dintre clienti poseda informatia „ arrivalTime ” ( timpul din simulare la care ajunge la o casa ) . Cand acest timp este egal cu timpul real al simularii , clientul vine si se aseaza la o casa . Modul in care se aseaza clientii la case a fost descris in partea de Alogoritmi ( clinetul isi alege coada cea mai scurta ) . Pentru fiecare client care pleaca , contorizam deasemenea si timpul cat a fost nevoit sa astepte pana sa-i inceapa procesarea produselor acestuia . La finalul simularii se culeg datele care au fost pana in acel moment si se calculeaza valorile necesare utilizatorului . Daca se intampla ca simularea sa se termine si sa mai existe clienti la case , am ales ca pe acei clienti aplicatia sa ii ia in considerare . Ora de varf este deasemenea calculata pe baza numarului de clineti din magazin la un anumit moment de timp . Aceasta informatie este recalculata in fiecare secunda .

**Concluzii :**

Implementarea acestui mini-proiect a fost aleasa in felul acesta deoarece este cea mai apropiata de modul in care lucreaza cozile in viata reala . Nu intamplator , acesta este unul dintre cele mai bune moduri de a modela aceasta aplicatie , deoarece , din experienta oamenilor , acestia au dezvoltat in timp un model pentru a reduce timpul de asteptare cat mai mult pentru a putea creste productivitatea .

Probabil in timp , oamenirea va crea alte modele de cozi , care sa reduca si mai mult timpul de asteptare . Tehnologia este in considera dezvoltare si eu consider ca singurul lucru de care depinde implementarea noilor modele de cozi este timpul .

Revenind la mini-proiectul prezentat , mi s-a parut o foarte buna aplicatie pentru dobandirea cunostintelor despre firele de executie a calculatoarelor si utilizarea lor pe viitor in aplicatii mai complexe .

**Dezvoltari ulterioare**

Momentan , acest model de coada este folosit si in viata reala , dar cum spuneam si in sectiunea precedenta , cu timpul se vor dezvolta noi metode de a facilita activitatile oamenilor , fara necesitatea unei cozi de asteptare .

Pentru aceasta aplicatie , algoritmul de aranjare a clientilor la cozi poate fi imbunatatit prin calcularea cozii optime in functie de timpul mediu de asteptare la coada respectia .

O alta facilitate importanta ar fi modul de gestionare al clientilor , in special ar trebui tratat cazul in care simularea se opreste si inca exista clienti la cozi . Timpii intermediari ai acestora ar trebui luati in considerare la calcularea valorilor finale .

**Cunostinte dobandite**

Pe parcursul dezvoltarii aplicatiei am fost nevoit sa fac studii despre functionarea firelor de executie ( Thread-urilor ) si atat modului lor de lucru individual cat si despre sincronizarea lor pentru lucrul in grup .

Pe langa acestea , mi-am dezvoltat cunostiintele de grafica am invatat moduri de afisare a unor informatii in timp real utilizatorilor prin crearea de grafice si log-uri de evenimente .

**Bibliografie**

http://adi.ro/pub/scoala/An%20III/Sem%20I/Java/FireEx.pdf

https://www.ms.sapientia.ro/~manyi/teaching/oop/oop\_romanian/curs6/curs6.html

<http://inf.ucv.ro/documents/tudori/laborator8_53.pdf>

<http://www.javapractices.com/topic/TopicAction.do?Id=62>

[www.stackoverflow.com](http://www.stackoverflow.com/)

<http://docs.oracle.com/javase/tutorial/>