OrderManagement

Documentație

Beltechi Marius Gabriel

Grupa 30227 An 2 semestrul 2

Cuprins

1.Obiectiv

Obiectivul acestui proiect a fost pentru procesarea comenzilor clienților pentru un depozit.
Bazele de date relaționale sunt utilizate pentru a stoca produsele, clienții și comenzile. În plus, cererea
utilizează (minim) următoarele clase:

- Clase de model - reprezintă modelele de date ale aplicației (de exemplu, comandă, client, produs)
- Business Logic clase - conține logica aplicației (de exemplu, OrderProcessing,
WarehouseAdmin, ClientAdmin)
- Clase de prezentare - clase care conțin interfața grafică a utilizatorului
- Clase de acces la date - clase care conțin accesul la baza de date

Pe langa partea tehnica putem gasi si o interfata grafica “User Friendly” care poate fi utilizata de catre orice utilizator.

2. Studiul probelemei

Analiza domeniului aplicației, determinați structura și comportamentul clasei sale și desenați o extensie
Diagrama clasei UML.. Implementarea clasele de aplicații. Utilizarea tehnici de reflecție pentru a crea o metodă care primește o listă de obiecte și generează antetul lui
tabel prin extragerea prin reflecție a proprietăților obiectului și apoi popularea tabelului cu valorile lui
elementele din listă.
JTable createTable (obiecte listă <Object>);Implementați un sistem de programe de utilitate pentru raportare, cum ar fi: stocuri, totaluri, filtre etc.

Conectivitatea bazei de date Java (JDBC) este o interfață de programare a aplicațiilor (API) pentru limbajul de programare Java, care definește modul în care un client poate accesa o bază de date. Este o tehnologie de acces la date bazată pe Java utilizată pentru conectivitatea

bazei de date Java. Face parte din platforma Java Standard Edition, de la Oracle Corporation. Oferă metode de interogare și actualizare a datelor într-o bază de date și este orientată spre baze de date relaționale. O punte JDBC-ODBC permite conectarea la orice sursă de date accesibilă cu ODBC în mediul gazdă al mașinii virtuale Java (JVM).

JDBC permite ca mai multe implementări să existe și să fie utilizate de aceeași aplicație. API oferă un mecanism pentru încărcarea dinamică a pachetelor corecte Java și înregistrarea acestora în JDBC Driver Manager. Managerul de drivere este folosit ca o fabrică de conexiuni pentru crearea conexiunilor JDBC. Conexiunile JDBC acceptă crearea și executarea declarațiilor. Acestea pot fi instrucțiuni de actualizare, cum ar fi SQL CREATE, INSERT, UPDATE și DELETE, sau pot fi instrucțiuni de interogare, cum ar fi SELECT. În plus, procedurile stocate pot fi invocate printr-o conexiune JDBC. JDBC reprezintă instrucțiuni utilizând una din următoarele clase: Instrucțiune - instrucțiunea este trimisă la serverul de bază de date de fiecare dată. PreparedStatement - instruciunea este stocată în memoria cache și apoi calea de execuție este predeterminată pe serverul bazei de date, permițându-i să fie executată de mai multe ori într-o manieră eficientă. CallableStatement - folosit pentru executarea procedurilor stocate în baza de date.

3.Implementare

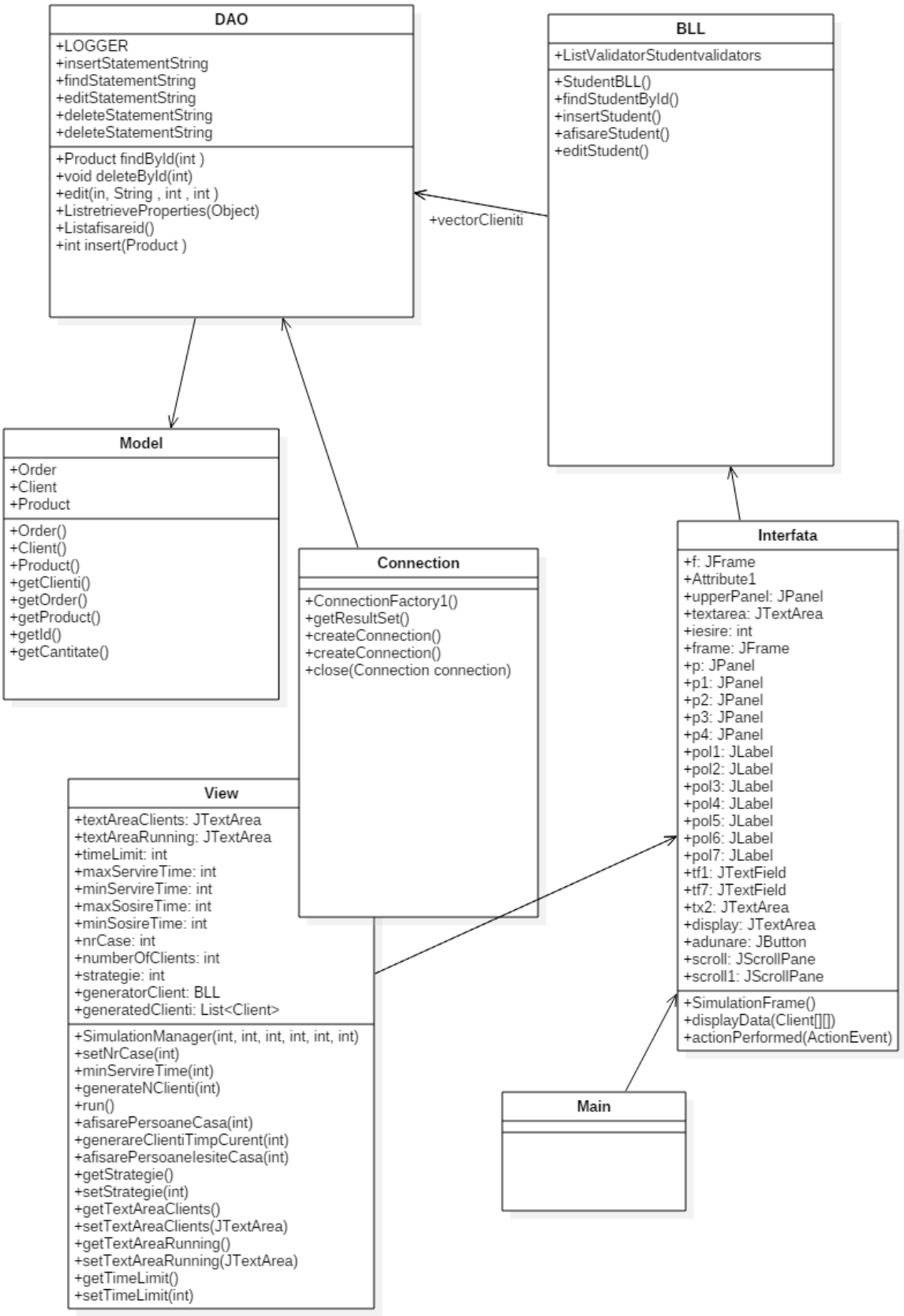
3.1

Diagrame UML

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software.

UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.

Prima versiune de UML, UML 1.0, a apărut în anul 1990 ca reacție a numeroaselor limbaje de modelare propuse pe piață. UML îi are ca fondatori pe Grady Booch, Ivar Jacobson și James Rumbaugh, așa numiții „cei trei *Amigos*”. Ei au dezvoltat limbajul bazându-se inclusiv pe limbaje de modelare deja existente, însă incomplete ca gamă de funcționalități. Printre acestea se numără și OOSE, RDD, OMT, OBA, OODA, SOMA, MOSES și OPEN/OML.


DAO
+LOGGER
+insertStatementString
+findStatementString
+editStatementString
+deleteStatementString
+deleteStatementString
+Product findById(int)
+void deleteById(int)
+edit(in, String , int , int)
+ListretrieveProperties(Object)
+Listafisareid()
+int insert(Product)
BLL
+ListValidatorStudentvalidators
+StudentBLL()
+findStudentById()
+insertStudent()
+afisareStudent()
+editStudent()
+vectorClieniti
Model
+Order
+Client
+Product
+Order()
+Client()
+Product()
+getClienti()
+getOrder()
+getProduct()
+getId()
+getCantitate()
Connection
+ConnectionFactory1()
+getResultSet()
+createConnection()
+createConnection()
+close(Connection connection)
Interfata
+f: JFrame
+Attribute1
+upperPanel: JPanel
+textarea: JTextArea
+iesire: int
+frame: JFrame
+p: JPanel
+p1: JPanel
+p2: JPanel
+p3: JPanel
+p4: JPanel
+pol1: JLabel
+pol2: JLabel
+pol3: JLabel
+pol4: JLabel
+pol5: JLabel
+pol6: JLabel
+pol7: JLabel
+tf1: JTextField
+tf7: JTextField
+tx2: JTextArea
+display: JTextArea
+adunare: JButton
+scroll: JScrollPane
+scroll1: JScrollPane
+SimulationFrame()
+displayData(Client[][])
+actionPerformed(ActionEvent)
View
+textAreaClients: JTextArea
+textAreaRunning: JTextArea
+timeLimit: int
+maxServireTime: int
+minServireTime: int
+maxSosireTime: int
+minSosireTime: int
+nrCase: int
+numberOfClients: int
+strategie: int
+generatorClient: BLL
+generatedClienti: List<Client>
+SimulationManager(int, int, int, int, int, int)
+setNrCase(int)
+minServireTime(int)
+generateNClienti(int)
+run()
+afisarePersoaneCasa(int)
+generareClientiTimpCurent(int)
+afisarePersoanelesiteCasa(int)
+getStrategie()
+setStrategie(int)
+getTextAreaClients()
+setTextAreaClients(JTextArea)
+getTextAreaRunning()
+setTextAreaRunning(JTextArea)
+getTimeLimit()
+setTimeLimit(int)
Main

3.2 Clase

Clasa Interfata:

Aceasta clasa este granita dintre interfata grafica si programul propriu-zis, unde se petrece interactiunea utilizator-sistem de calcul.

Clasa Main:

Aici este clasa cu main-ul in care ii spunem programului ce sa execute. Aici doar am realizat deschiderea interfetei grafice.

Clasa Client:

In aceasta clasa am declarat clientul avand un id,nume,adresa,email si varsta. Am pus Setter si Getter pe fiecare dintre acestea pentru a putea sa fie mai usor de folosit

Clasa Product:

In aceasta clasa am declarat clientul avand un id,nume,pret si cantitate. Am pus Setter si Getter pe fiecare dintre acestea pentru a putea sa fie mai usor de folosit.

Clasa Order:

In aceasta clasa am declarat clientul avand un id,client,produs si cantitate. Am pus Setter si Getter pe fiecare dintre acestea pentru a putea sa fie mai usor de folosit.

Clasa StudentDAO:

In aceasta clasa am facut cate o metoda pentru adaugare client, stergere client, afisare clienti, modificare clienti si gasire client. Pe lange aceasta am mai realizat si conexiunea la tabelul din baza de date pentru a exrtrage informatiile necesare din acesta.

Clasa ProductDAO:

In aceasta clasa am facut cate o metoda pentru adaugare produs, stergere client, produs, modificare produs si gasire produs. Pe lange aceasta am mai realizat si conexiunea la tabelul din baza de date pentru a exrtrage informatiile necesare din acesta.

Clasa OrderDAO:

In aceasta clasa am facut cate o metoda pentru adaugare comanda. Pe lange aceasta am mai realizat si conexiunea la tabelul din baza de date pentru a exrtrage informatiile necesare din acesta.

Clasa StudentBll

In aceasta clasa am preluat metodele din StudentDAO si am valitat email-ul si varsta pentru fiecare client.

Clasa Controller

Aceasta clasa este granita dintre interfata grafica si programul propriu-zis, unde se petrece interactiunea utilizator-sistem de calcul. Aici am avut 3 butoane pentru alegerea tabelului

Clasa View

Aceasta clasa este granita dintre interfata grafica si programul propriu-zis, unde se petrece interactiunea utilizator-sistem de calcul. Aici am avut 3 cazuri separare de view in functie de table. Aici am apelat metodele din BLL in functie de fiecare caz(daca am avut de exemplu de adaugat un client am apelat insert din StudentBLL).

3.3 Metode

Metode utilizate in clasa ***Client***

Aici am declarat structura unui client. El este format din id, nume, adresa,email si varsta. Pentru fiecare dintre acestea am realizat o metoda de setare a id-ului, timp servire si sosire si una de extragere a acesteia.

```
public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getAddress() {
		return address;
	}

	public void setAddress(String address) {
		this.address = address;
	}

	public int getAge() {
		return age;


	}
```

Pe langa acestea am mai facut o metoda toString pentru a returna id-ul.

Metode utilizate in clasa DAO

In aceasta clasa am creat cinci metode: una pentru afisare Clienti, inserare Client, cautare Client, adaugare Client, editare Client. La fiecare dintre aceste metode am folosit PreparedStatement in functie de lucrul pe care vrem sa il luam sau modificam din baza de date. În sistemele de gestionare a bazelor de date (DBMS), o instrucțiune pregătită sau o instrucțiune parametrizată este o caracteristică folosită pentru a executa aceleași declarații de bază de date sau similare în mod repetat, cu eficiență ridicată. În mod obișnuit, cu instrucțiuni SQL, cum ar fi interogări sau actualizări, instrucțiunea pregătită are forma unui șablon în care anumite valori constante sunt înlocuite în timpul fiecărei execuții. Am realizat conexiunea la baza in fiecare metoda si am folosit ResultSet pentru manipularea datelor din tabela. Un set de rezultate SQL este un set de rânduri dintr-o bază de date, precum și metadate despre interogare, cum ar fi numele coloanelor și tipurile și dimensiunile fiecărei coloane. În funcție de sistemul bazei de date, numărul de rânduri din setul de rezultate poate sau nu să fie cunoscut. De obicei, acest număr nu este cunoscut în față, deoarece setul de rezultate este construit pe-the-fly.

Un set de rezultate este efectiv un tabel. Clauza ORDER BY poate fi utilizată într-o interogare pentru a impune o anumită condiție de sortare pe rânduri. Fără această clauză, nu există nicio garanție în ceea ce privește ordinea în care rândurile sunt returnate.

```
public static Product findById(int studentId) {
		Product toReturn = null;

		Connection dbConnection = ConnectionFactory1.getConnection();
		PreparedStatement findStatement = null;
		ResultSet rs = null;
		try {
			findStatement =
dbConnection.prepareStatement(findStatementString);
			findStatement.setLong(1, studentId);
			rs = findStatement.executeQuery();
			rs.next();

			String name = rs.getString("name");
			int pret = rs.getInt("pret");
			int cantitate = rs.getInt("cantitate");
			toReturn = new Product(studentId, name, pret,cantitate);
		} catch (SQLException e) {
			LOGGER.log(Level.WARNING,"productDAO:findById " +
e.getMessage());
		} finally {
			ConnectionFactory1.close(rs);
			ConnectionFactory1.close(findStatement);
```

```
                ConnectionFactory1.close(dbConnection);
            }
            return toReturn;
            }
```

Metode utilizate in clasa BLL

In aceasta clasa am creat cinci metode: una pentru afisare Clienti, inserare Client, cautare Client, adaugare Client, editare Client preluate din DAO. Am mai folosit o lista de validare pentru a verifica daca clientul are varsta si email-ul necesar.

```
public Student findStudentById(int id) {
            Student st = StudentDAO.findById(id);
            if (st == null) {
                throw new NoSuchElementException("The student with id =" + id + "
was not found!");
            }
            return st;
            }
```

Metode utilizate in clasa Controller

In aceasta clasa am creat interfata grafica. Am pus trei butoane, unul pentru fiecare table. La apasarea unuia se deschide o noua interfata numita view.

Metode utilizate in clasa View

In aceasta clasa am creat cate un buton pentru adaugare,inserare,editare, afisare cautare. La apasarea unui buton se executa instructiunea corespunzatoare din BLL. De exemplu daca apasam “Afisare clienti” se va apela metoda de afisare din ClientiBLL si ne va genera un table cu toti clientii din baza de date. Aici am folosit reflection pentru generarea datelor din table. Reflecția este mecanismul prin care Java expune caracteristicile unei clase în timpul rulării, permițând programelor Java să enumere și să acceseze ca obiecte metode, câmpuri și constructori de clasă. Cu alte cuvinte, există oglinzi bazate pe obiecte care reflectă modelul de obiect Java și puteți utiliza aceste obiecte pentru a accesa caracteristicile unui obiect utilizând construcții API runtime în loc de construcții de limbaj de compilare. Fiecare instanță a obiectului are o metodă getClass (), moștenită de la java.lang.Object, care returnează un obiect cu reprezentarea runtime a clasei obiectului respectiv; acest obiect este o instanță a clasei java.lang.Class, care la rândul său are metode care returnează câmpurile, metodele,

constructorii, superclasele și alte proprietăți ale clasei respective. Puteți utiliza aceste obiecte de reflecție pentru a accesa câmpurile, pentru a invoca metode sau instanțiate instanțe, toate fără a avea dependențe de compilare în funcție de acele caracteristici

3.4 GUI

Interfarta grafica sau GUI este o interfata cu utilizatorul bazata pe un sistem de afisaj ce utilizeaza elemente grafice. Interfata grafica este numit sistemul de afisaj grafic-vizual pe un ecran. Situate functional intre utilizator si dispositive electronice cum ar fi computere Pentru a prezenta toate informatiile si actiunile disponibile, un GUI ofera pictograme si indicatori vizuali, in contrast cu interfetele bazate pe text, care ofera doar nume de comenzi.

Interfata grafica are urmatoarele component: frame, panel, label,textfield, buton.

Frame este “rama” in care se adauga toate elementele de care avem nevoie pentru program.

Panel sunt cele patru panouri propriu-zise. Am facut un panou pentru primul polinom, unul pentru al doilea, altul pentru rezultatul polinomului si unul pentru operatii. In fiecare panou am adaugat label, textfield si buton representative acestuie.

Label-urile le-am folosit pentru a arata ce trebuie sa insereze utilizatorul pentru a face o anumita operatie. De exemplu, la polinom1 sa introduca un polinom in textfield-ul respective.

TextField este practice o casuta text in care utilizator introduce un text, iar programul citeste acest text si il modifica conform intructiunilor din spate. Am pus trei textField-uri, doua pentru cele doua polinoame si unul pentru rezultatul lor. De exemplu tf1 si tf2 sunt textField-urile pentru cele doua polinoame introduse de utilizator, iar tf3 este cel pentru afisarea polinomului rezultat.

Butoanele executa o anumita instructiune in momentul in care sunt apasate.

Am pus patru butoane, cate unul pentru fiecare operatie. Pentru fiecare button, s-a făcut o metodă nouă care implică ActionEvent. Astfel, de fiecare dată când are loc o acțiune la un buton, de exemplu a fost apăsat, ActionEvent-ul denumit e, transmite informația la ActionListener care așteaptă astfel de informații. Apoi, au loc evenimentele ce se afla în metodă respective.

TextField = spatii dreptunghiulare in care se pot introduce date de la tastatura. Dar pe langa asta pot fi folosite si pentru a afisa rezultatul fara a se putea introduce date de la tastatura.

Un obiect JLabel poate afișa text, imagine sau ambele. Puteți specifica unde în zona de afișare a etichetei conținutul etichetei este aliniat prin setarea alinierii verticale și orizontale. Implicit, etichetele sunt centrate pe verticală în zona de afișare a acestora. Etichetele numai cu text sunt aliniate la marginea de vârf, în mod implicit; numai etichetele cu imagini sunt centrate pe orizontală, în mod implicit. Spre deosebire de câmpurile de text și de multe alte elemente de control, JLabels sunt transparente în mod implicit: conținutul parental este vizibil în locuri care nu sunt acoperite de text și pictograme, iar setarea culorii de fundal nu are efect. JLabel poate fi opacă prin setOpaque (adevărat).

\

4.Concluzii

In concluzie sunt de parere ca in acest proiect, am invatat sa proiectez si sa utilizez o interfata grafica, am aprofundat mai bine limbajul JAVA, implementarea paradigmelor OOP.

5.Bibliografie

1. Youtube
2. Wikipedia
3. https://beginnersbook.com/
4. https://examples.javacodegeeks.com