

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

**Order management**

**Documentație**

**ASSIGNMENT 3**

**Nume si prenume: Stecko Daiana**

**Grupa 302210 | An 2 semestrul 2**

**Profesor laborator: Mitrea Dan**

**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:**

Obiectivul temei trei a fost sa proiectam in Java o aplicatie care sa simuleze comenzile clientilor. Avem produse si client, iar clientii fac comenzii si acele comenzi sunt facute din produse. Acest program trebuie sa contina minim 3 tabele, si anume Client, Product si Order. Programul trebuie sa aiba date de intrare, le introduce noi si sa isi realizeze operatiile specificate (insert, delete, update etc). Programul trebuie sa aiba o legatura la o baza de date SQL cu care sa opereze.

1. **Analiza problemei, modelare, scenarii, cazuri de utilizare:**

Modelarea problemei a fost realizata in general dupa exemplul din prezentare. Fiecare tabel are asignata o clasa care are ca atribute coloanele tabelului. Modelarea a fost facuta pe nivele, cum a fost cerut, astfel incat sunt 4 nivele de executie, fiecare fiind reprezentat printr-un pachet:

1. DataAccess (contine clasele: ConnectionFactory, Queier), care contine clase necesare cu tabelele din baza de date.
2. Model (contine clasele: Client, Product, Order), unde sunt implementate query-urile pentru fiecare actiune din fiecare clasa.
3. BussinessLogic (contine clasele: Creator, Destroyer, Selector, Updater), care se ocupa cu definirea concreta a actiunilor pentru fiecare clasa.
4. Presentation, aceasta contine clasele care se ocupa de parsarea fisierului de intrare si extragerea datelor de intrare si apleleaza metodele din clasele aferente actiunii dorite si mai avem main-ul proiectului,

**Scenarii/ Cazuri de utilizare**

Cazul de success:

Pentru ca instructiunile din fisierul de intrare sa se execute corect, acestea trebuie sa respecte intocmai forma fiecarei instructiuni:

1. Insert:
2. Client: “Insert client: nume, prenume, adresa”
3. Product: “Insert product: nume, cantitate, pret”
4. Order: “Order: nume prenume, nume\_produs, cantitate ”

“nume prenume” se refera la numele si prenumele clientului.

1. Delete:
2. Client: “Delete client: nume prenume”
3. Product: “Delete product: nume”
4. Report:
5. Client: “Report client”
6. Product: “Report product”
7. Order: “Report order”

A NU SE INSERA de doua ori produse sau clienti cu acelasi nume!

Cazuri de esec:

1. Lipsa datelor de intrare, cale incomplete catre acesta.
2. Daca nu se respecta forma exacta pentru fiecare instructiune, va aparea o eroare sau pur si simplu va fi ignorata comanda.
3. Daca se adauga de exemplu la delete si adresa clientilor sau cantitatea/pretul produselor, aceasta instructiune va fi ignorata
4. Daca un client comanda o cantitate dintr-un produse mai mare decat stocul curent al acelui produs in magazin.

**3.** **Proiectare (decizii de proiectare, diagrame UML, structure de dare, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)+ 4. Implementare**

Proiectarea claselor a fost facuta dupa modelul prezentat mai sus. Astfel, fiecare pachet are un rol prestabilit de dinainte si in proiectare acestora, nu s-a sarit peste nivele. Fiecare fiind in corelatie cu nivelul anterior.

**Pachetul BusinessLogic:**

Avem clasa Creator care ne creaza un queier si apeleaza metoda insert pe querier.

Avem clasa Destroyer care ne creaza un querier si apeleaza metoda delete pe querier.

Avem clasa Selector, aceasta avand doua clase, un care poate sa selecteze toate sau poate sa umple o tabela.

Avem clasa updater care ne creaza un queier si apeleaza metoda update pe querier.

**Pachetul DataAccess:**

In acest pachet avem doua clase. Prima clasa se numeste Connection Factory. Aceasta ne ajuta sa ne conectam la baza de date si creaza un o singur instanta pentru baza de date. Practic aceasta adauga un nou obiect intr-un obiect care exista deja si daca noi incercam sa afisam acel obiect create ne va returna primul obiect si nu cel inserat de noi.

Cea de a doua clasa se numeste Querier si se creaza cu o clasa T, aceasta clasa T poate fi ori client,ori produs, ori productOrder, practic ii o reprezentare virtuala a ce avem noi in baza de date. In metoda ‘ALL’ toate entitatile din baza noastra de date, cele de tip T, cream un array list nou in care punem rezultatele, intitializam conexiunea. Statementul primeste de fapt statementul pe care vrem sa-l rulam in baza de date. Dupa care returnam toate randurile pe care le selectam. Vom creea o instanta noua de obiect T care poate fi client, produs, order si pentru fiecare field o sa ne selectam valoarea de numele f.getName. Dupa vom creea un propertyDescriptor care o sa ne descrie proprietatea dorita, iar la final am pus o exceptie care va fi “aruncata” in cazul unei erori. In final am inchis conexiunile.

In metoda ‘INSERT’ am facut din nou o conexiune, dupa care am apelat functia buildSqlQuery, care o sa ne ia un obiect si ne va genera un insertStatement care de fapt insereaza toate field-urile. Dupa am facut un for, am pornit cu f de la 1 si propertyDescriptor primeste field-urile de la pozitia f. Pentru fiecare field, in cazul in care-i de tipul int, il vom seta ca si un int, iar daca-i de tipul string o sa-l setam ca string. Dupa care vom rula statementul execute, care nu o sa returneze nimic. In caz de greseli primim alerta. Iar la final am pus o exceptie care va fi “aruncata” in cazul unei erori. In final am inchis conexiunile.

In metoda ‘UPDATE’, mi-am construit updateQuery si am initializat statementul. Dupa care am facut un for, prin care o sa selectam toate filed-urile din clasa noastra, o sa luam propertyDescriptor-urile acestor field-uri, adica inseram din noua la “?”, datele noastre. Update-ul are un id la final. Mai vom selecta un propertyDescriptor pe field-ul de id sim ai setam la return in de ultimul “?”, adica invocam getter-ul pentru id. Iar la final am pus o exceptie care va fi “aruncata” in cazul unei erori. In final am inchis conexiunile.

Iar la metoda ‘DELETE’ am facut in mod similar cu cele prezentate mai sus.

**Pachetul presentation:**

Acest pachet este interfata noastra. In acest pachet avem trei clase mari. Prima clasa se numeste GUI, aici o sa avem un root unde adaugam niste tab-uri. La tab pane am adaugat cel trei Gui-luri din celelalte doua clasen unul pentru client, produs, order. Dupa care am create tab-urile si le -am adaugat. Am facut o functie care sa adauge la un listener si care va da refresh la date de fiecare data cand orderTab-ul este selectat. Acelasi lucru am facut si la order.

Dupa avem clasa GUIWindowClient, in care avem un constructor unde vom face aplicatia. In ceasta clasa n e-am declarant butoanele, create, delete, edit, retrieve. Tot aici am facut un borderpane care reprezinta de fapt punctul noastru principal din interfata. Pentru root am mai facut un hbox, o metoda de get si set. Aici mai avem un textfield pe care -l folosim sa adaugam date in tabela noastra, care-I definite mai jos in cod. Am dat Client ca fiind un generic, iar continutul tabelei va fi de tip client, dupa care am initializat interfata.

In metoda ‘INIT’, la clienttableview setam resize\_policy la coloane sa ocupe tot spatial posibil pentru a nu avea spatii goale. Apelam refresh table care o sa ne bage datele in tabela. Dupa care se va apela de fiecare data cand selectam un lucru din tabela si va pune pe acel lucru in text field. Cand vom apasa pe create vom lua ce-I in textfield-uri unui client nou si vom chema metoda run de pe creator pentru client class si clientul noastru, iar in final vom da refresh. Iar la final am pus o exceptie care va fi “aruncata” in cazul unei erori.

Update.setonaction ne face exact acelasi lucru, doar ca ce difera ii faptul ca cu updater si se cheama clientul nostru selectat in tabela. Iar la final am pus o exceptie care va fi “aruncata” in cazul unei erori.

La delete la fel se cheama clientul current si se ruleaza destroyer.

In cadrul acestui pachet mai avem si clasa GUIWindowOrder. Mai avem si metoda init in care dam un refresh table, adica se duce si ia toate orderurile din baza de date si o sa le bage in tabela. Dupa care vom da un refresh data care va selecta toti si clientii si toate produsele , dupa care da clear la obsevableList. In aceasta clasa avem metoda init in care setam functia butonului de create, adica acest lucru o sa selecteze clientul, produsul, si o sa mai selecteze cantitatea in integer. Si atunci pentru fiecare produs din productArrayList, daca produsul care-l cautam este egal cu produsul la care suntem acum si daca cantitatea care vrem sa o luam este mai mare decat stocul acelui produs o sa “aruncam” o noua exceptie, iar in cazul in care avem cantitatea ceruta atunci scadem cantitatea comandata din stocul pe care-l avem si rula updater. Dupa ce am facut modificarile la produsul pe care vrem sa-l comandam vom crea un nou obiect order. In final rulam creator.

In finalul acestui pachet mai avem si clasa GUIWindowProduct. In aceasta clasa am declarant initial text field-urile pentru productName, productSupplier si productStock, de asemenea am mai facut un tableview pentru product. In metoda GUIWindowProduct am facut hbox, labe-rurile si si am setat padding.Mai avem si metoda init in care dam un refresh table, adica se duce si ia toate orderurile din baza de date si o sa le bage in tabela. Dupa care folosim getSelectionModel pentru property. Dupa care am facut un if daca noua selectie este diferita de null atunci folosim settext pentru name, supplier, stock, dupa refresh table. In create facem un try cach adaugam productname si productsupplier,si vom setam functia butonului product. Dupa care vom apela creator pe produs p si vom da refresh table. Iar la final am pus o exceptie care va fi “aruncata” in cazul unei erori. Update.setonaction ne face exact acelasi lucru, doar ca ce difera ii faptul ca cu updater si se cheama clientul nostru selectat in tabela. Iar la final am pus o exceptie care va fi “aruncata” in cazul unei erori. La delete la fel se cheama clientul current si se ruleaza destroyer.

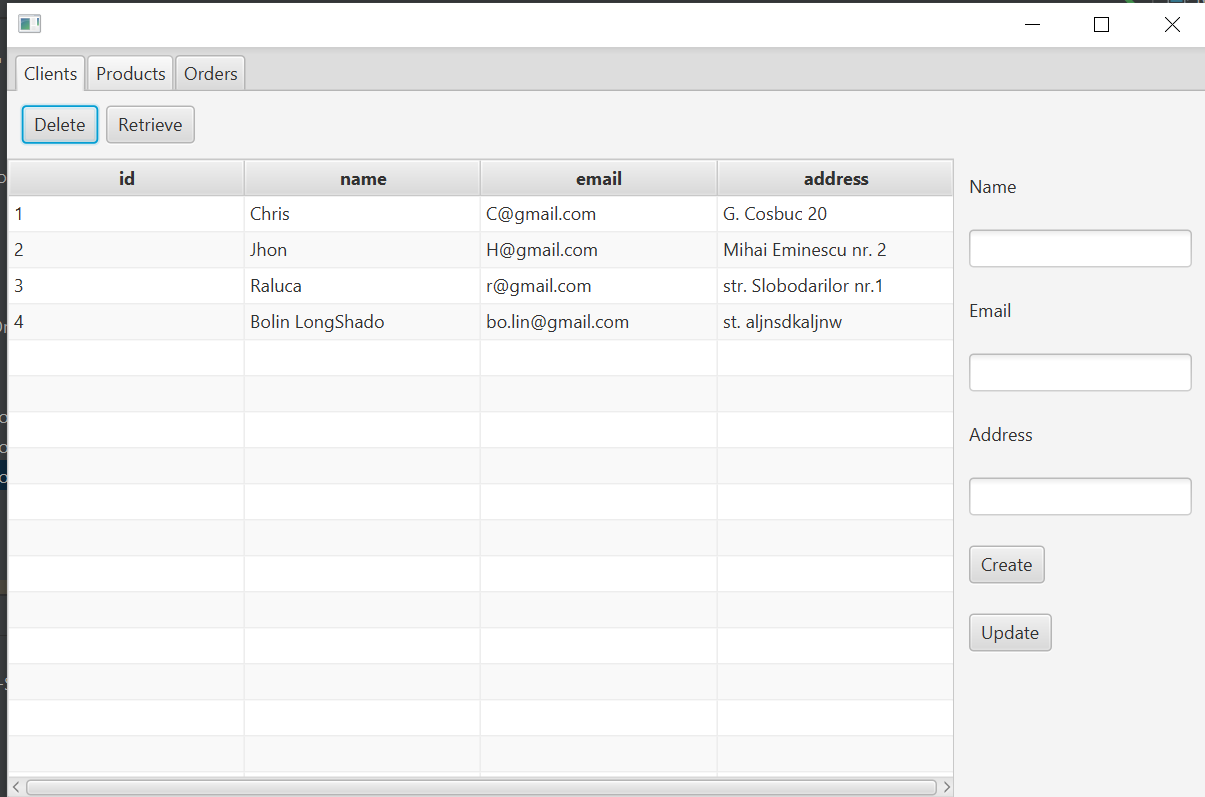
**Pachetul Model:**

In acest pachet avem trei clase. Prima clasa care se numeste Client. In aceasta clasa avem id, name, email, address. Tot aici mai facut gettere si setere pentru fiecare.

Cea de a doua clasa se numeste Product. La nivelul acestei clase avem definit id-ul, name, supplier si stock. Si am creat cate un getter si seter pentru fiecare.

In cea de a treia clasa, Product order, la fel am definit id, client, product, quantity, plus am facut gettere si setare pentru fiecare.

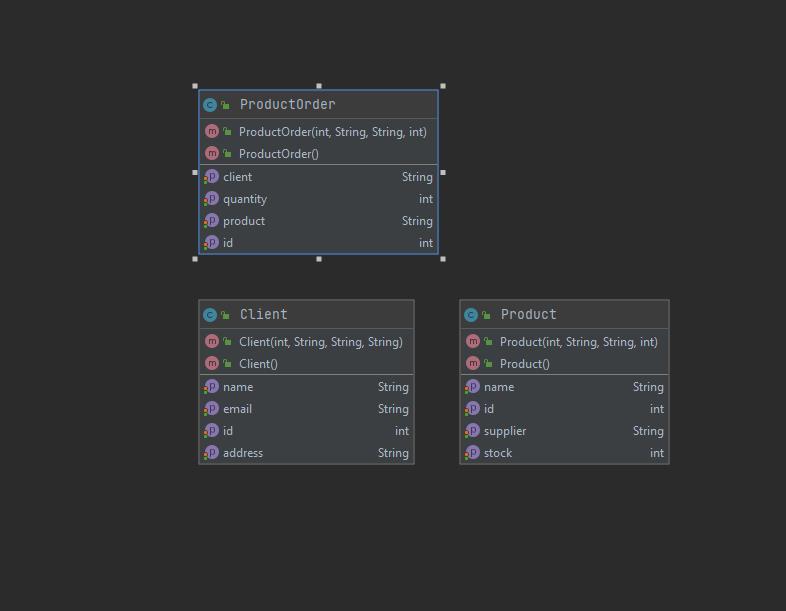
**Interfata:**



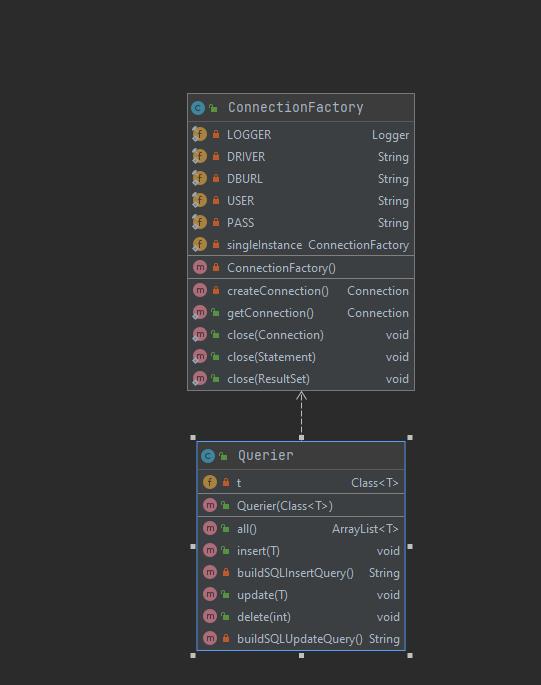
Unified Modeling Language (adica prescurtat UML) este un limbaj standard pentru descrierea de [modele](https://ro.wikipedia.org/w/index.php?title=Modelare_orientat%C4%83_pe_obiect&action=edit&redlink=1) și [specificații](https://ro.wikipedia.org/w/index.php?title=Specifica%C8%9Bie&action=edit&redlink=1) pentru [software](https://ro.wikipedia.org/wiki/Software). UML are la baza dezvoltarea pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte).In cazul cestui proiect am folosit diagrama te tip clase, diagramele de clasă sunt folosite pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte, și a legăturilor concrete dintre acestea.

Am generat o diagrama pentru fiecare pachet in functie legaturile pe care le are.

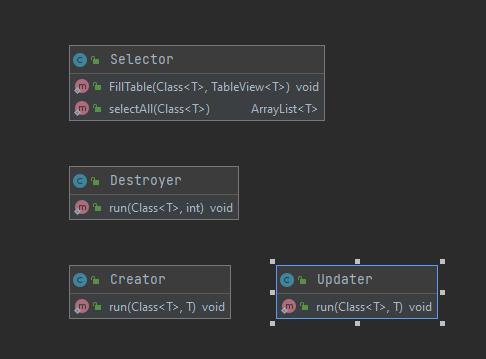
Avem mai jos pachetului Model care contine clasele Client, Product si ProductOrder.



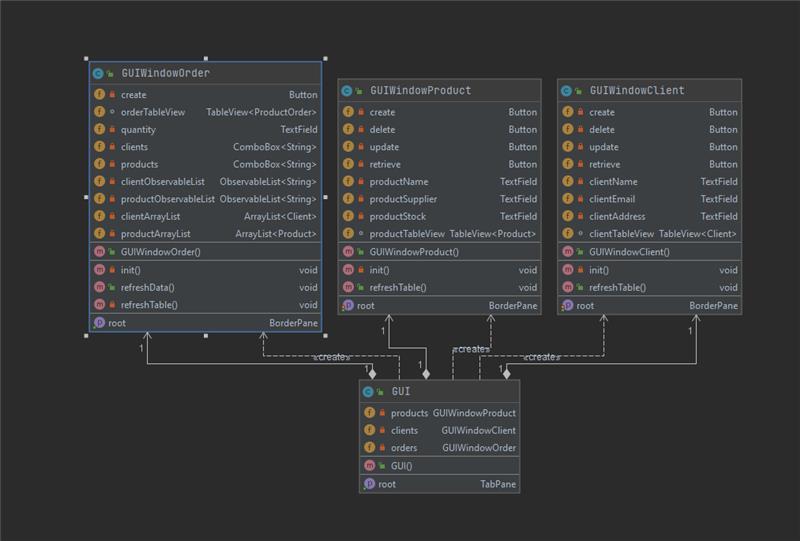
Dupa care am generat pachetul DataAccess care contine ConnectionFactory si Querier.



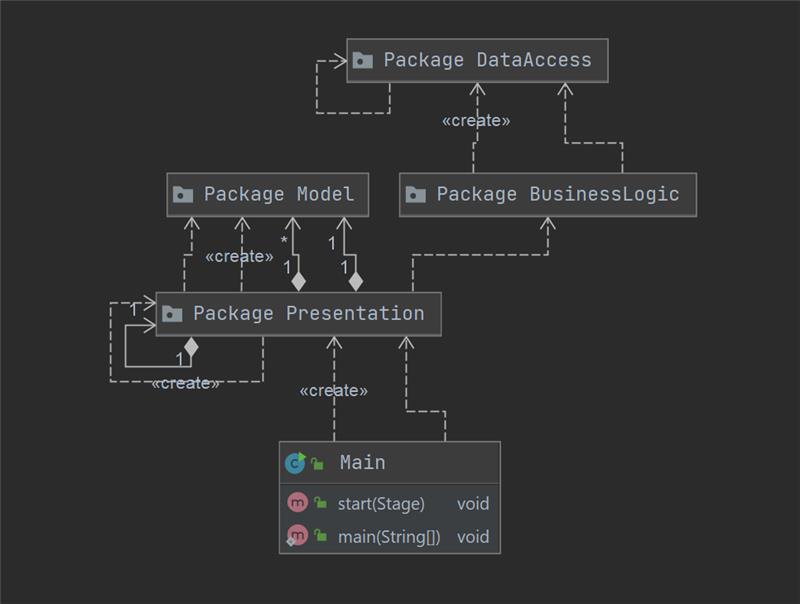
Dupa care am generat pachetul BusinessLogic care contine Creator, Destroyer, Selector si Updater.



Dupa care am generat pachetul Presentation care contine GUI, GUIWindowOrder, GUIWindowClient si GUIWindowProduct.



Pachetul de diagrame



**5. Rezultate**

Pentru testarea proiectului am fost nevoita sa realizez mai multe teste, am facut intial teste mai generale sa vad cum merge, l-am testat pe bucati, de la inceput cum scriam o parte testam sa vad daca merge corect din motive de siguranta ca la final sa fie totul in regula. Rezultatele au fost corecte. Ca eroroi ce mi-a fost mai greu de rezolvat am intampinat pentru clasele din package-ul Model, clasa Client, Product, ProductOrder , dar am reusit sa rezolv problema si programul merge cu success.

**6.Concluzii**

Tema trei a fost primul meu proiect in care a trebuit sa lucrez cu bazele de data in Java. Partea de query-uri a fost destul de usoara, avand in vedere casemnestrul trecut am avut o materie separata pentru baze de date. Partea de cconectare a fost putin dificila, dar intr-un final, cu implicare si documentare, am reusit sa inteleg in mare parte ce se intampla si cum trebuie procedat la nivelul acestui proiect.

**7.Bibliografie**

Sursele principale de informatii au fost link-ul cu:

- https://www.jetbrains.com/help/idea/connecting-to-a-database.html

- https://stackoverflow.com/questions/44675448/intellij-idea-stackoverflowerror-on-build-project

- google.