

ORDER MANAGEMENT

Student: Blajan George-Paul

Grupa: 302210

CUPRINS

[***Obiectivul temei:*** 2](#_Toc69621155)

[***Analiza problemei:*** 2](#_Toc69621156)

[***Proiectare:*** 3](#_Toc69621157)

[***Implementare:*** 5](#_Toc69621158)

[***Rezultate:*** 11](#_Toc69621159)

[***Concluzii:*** 11](#_Toc69621160)

[***Bibliografie:*** 11](#_Toc69621161)

# ***Obiectivul temei:***

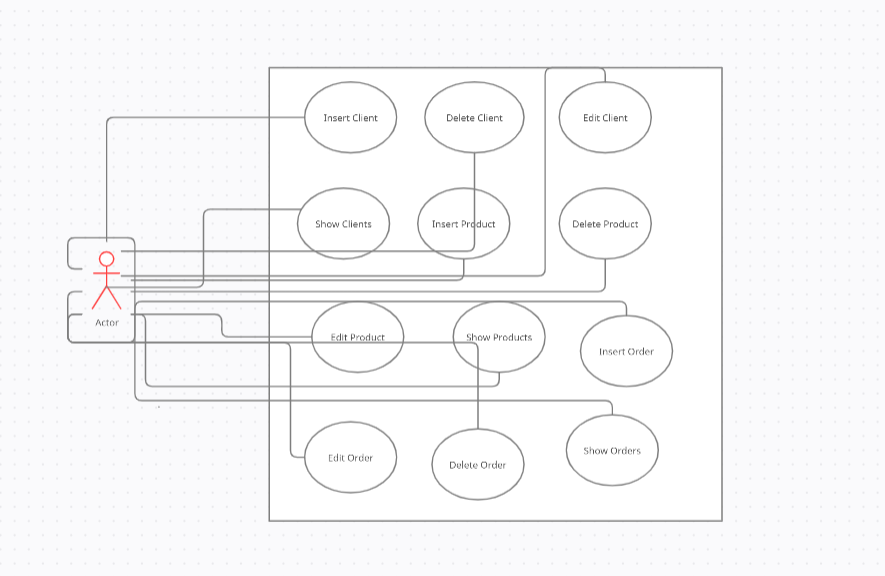
Obiectivul principal al acestei teme este realizarea unei aplicatii care sa simuleze un depozit de produse la care clientii pot face comenzi. Aplicatia este legata de o baza de date MySql prin procedee specifice OOP.

Pentru a indeplini obiectivul principal este nevoie sa fie indeplinite urmatoarelel obiective secundare: conexiunea la baza de date,realizarea interogarilor pe baza de date,insearare clienti,produse,comenzi in baza de date,sterge clienti,produse,comenzi din baza de date,editare clienti produse,comenzi si afisarea tuturor celor de mai sus.

# ***Analiza problemei:***

Cazurile de utilizare pentru aceasta aplicatie sunt reprezentate de operatiile care pot fi facute de aplicatie cum ar fi inserarea unui client,stergerea unui client,editarea unui client,inserarea unui produs,stergerea unui produs,editarea unui produs,inserarea unei comenzi,stergerea unei comenzi,editarea unei comenzi si afisarea celor de mai sus.

Diagrama use case:



# ***Proiectare:***

In acest proiect una dintre utilitatile OOP folosite a fost tehnica de reflection, **Reflection** reprezintă un mecanism ce permite unui program să-și examineze și să-și modifice structura in timpul rulării. La baza API-ului pentru **Reflection** stau următoarele clase :

•Class

•Constructor

•Method

•Field

•Annotation

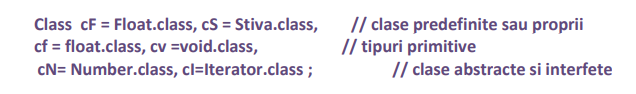
Fiecare dintre aceste clase abstractizează conceptul corespunzător numelui clasei. Implementarea internă a acestui mecanism este diferită de la un JDK la altul, dar în mare este bazată pe **metode native**.

Pentru fiecare clasă încărcată în maşina virtuală Java este creat automat câte un obiect de tip Class, cu informaţii despre clasa asociată (metadate). Obiectele Class sunt create automat şi pentru interfeţe, clase abstracte şi vectori intrinseci Java.

Prin ―reflecţie se înţelege obţinerea de informaţii despre o clasă sau despre un obiect în faza de execuţie . In plus, se pot crea şi modifica dinamic obiecte în faza de execuţie. Reflecţia este asigurată în principal de clasa numită Class.

O variabilă de tip Class conţine o referinţă la un obiect descriptor de clasă; ea poate fi iniţializată în mai multe feluri:

- Folosind cuvântul cheie class ca şi cum ar fi un membru public şi static al clasei sau tipului primitiv :

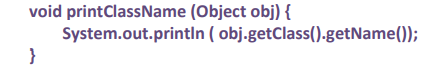


- Folosind metoda getClass() pentru o variabilă de orice tip clasă (metoda getClass() este moştenită de la clasa Object, deci există în orice clasă):

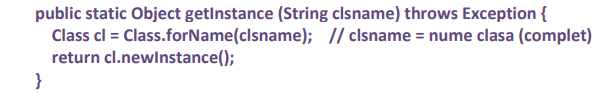


Clasa Class conţine metode care au ca rezultat numele clasei, tipul clasei (clasă sau interfaţă sau vector), tipul superclasei, clasa externă, interfeţe implementate, tipul obiectelor declarate în clasă, numele câmpurilor (variabilelor clasei), numele metodelor clasei, formele funcţiilor constructor s.a.

Clasa Class conţine metode care au ca rezultat numele clasei, tipul clasei (clasă sau interfaţă sau vector), tipul superclasei, clasa externă, interfeţe implementate, tipul obiectelor declarate în clasă, numele câmpurilor (variabilelor clasei), numele metodelor clasei, formele funcţiilor constructor s.a.

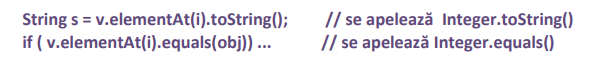


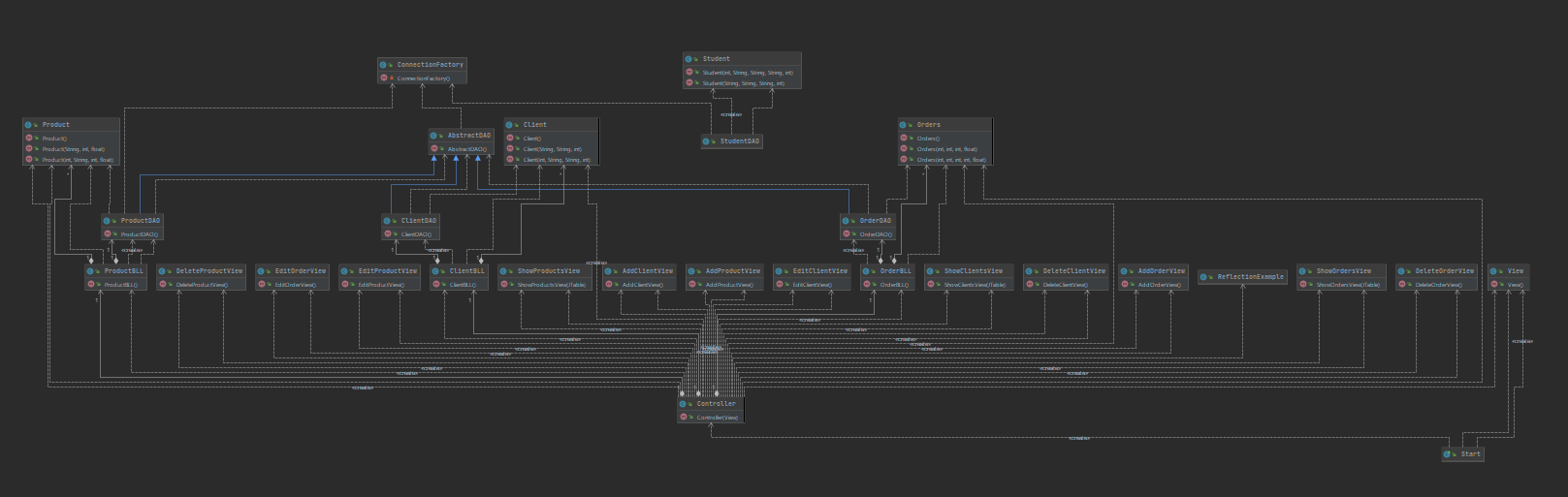
Crearea de obiecte de un tip aflat în cursul execuţiei dar necunoscut la compilare (cu condiţia ca tipul respectiv să fi fost definit printr-o clasă, iar fişierul de tip class să fie accesibil la execuţie) se poate face simplu dacă în clasă există numai un constructor fără argumente . Exemplu:



Prin reflecţie un asamblor de componente JavaBeans dintr-un mediu vizual poate să determine proprietăţile si metodele proprii unor obiecte, să modifice proprietăţile acestor obiecte şi să genereze apeluri de metode între obiecte. In rezumat, reflecţia permite operaţii cu clase şi cu obiecte necunoscute la scrierea programului, dar care pot fi determinate dinamic, în cursul execuţiei.

Dacă se apelează metode polimorfice pentru obiecte de tip necunoscut, atunci se va apela automat varianta definită în clasa respectivă, chiar dacă programatorul (si nici compilatorul) nu ştie care este tipul exact al obiectului. Exemplu:





# ***Implementare:***

Vom incepe descrierea proiectului cu interfata Validator.Interfata Validor are o singura metoda , Validare care primeste ca argument o variabila de tip generic T.

Avem de asemenea cateva clase care implementeaza interfata Validator, avem nevoie de ele evident pentru a indeplini functia de validare,se vor asigura ca datele primite de la interfata cu utilizatorul sunt corecte si nu fac rau in niciun fel bunei functionari a aplicatiei.

Clasa de validare sunt: ClientAgeValidator,ClientNameValidator,EmailValidator,IDValidator,PriceValidator,ProductNameValidator,QuantityOrderValidator,QuantityValidator.

Ckasa ClientAgeValidator se ocupa cu validarea varstei clientilor,in momentul cand un client este inserat in baza de date metoda Validate din clasa ClientAgeValidator este apelata iar varsta clientului este verificata pentru a corespunde limitelor impuse in prealabil.

Clasa ClientNameValidator se ocupa cu validarea numelui clientilor introdusi in baza de date,metoda Validate din aceasta clasa verifica ca numele clientului introdus sa nu fie contina cifre sau spatii pe baza unui regex predeterminat.

Clasa EmailValidator se ocupa de validarea email-ului clientilor inserati in baza de date , email-ul clientilor este comparat cu ajutorul unor obiecte Pattern si Matcher cu regexul construit in prealabil iar daca formatul email-ului nu corespunde cu cel al regexului se afiseaza o eroare.

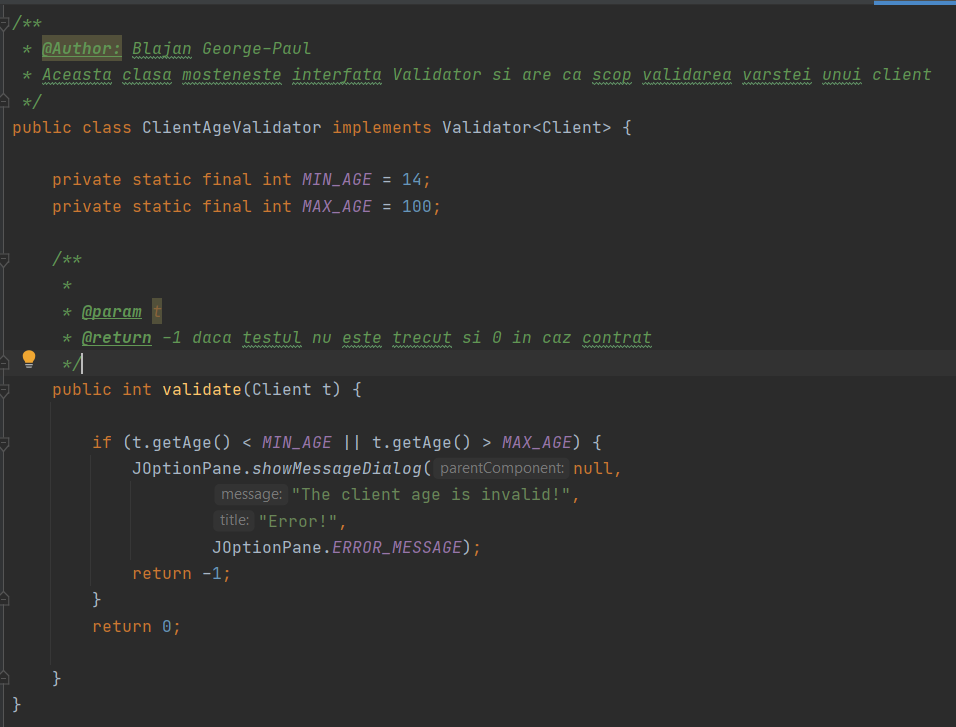
Clasa IDValidator se ocupa cu validarea ID – ului obiectelor inserate, daca ID-ul este un numar negativ sau nu este transmir ca un numar de catre utilizator si contine si litere este afisata o eroare.

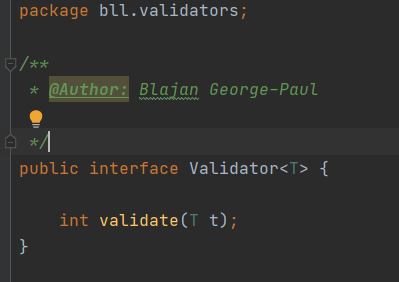
Clasa PriceValidator se ocupa cu validarea pretului asemenea celei de IDValidator,daca pretul este negativ sau nu corespunde formatului unui numar natural atunci este afisata o eroare.

Clasa ProductNameValidator se ocupa cu validarea numelui produsului inserat in baza de date,daca numele produsul contine cifre sau simboluri nepermise in interiorul acestuia este afisata o eroare,verificarea se face cu ajutorul obiectelor Pattern si Matcher.

Clasa QuantityOrderValidator se ocupa cu validarea cantitatii de produse comandate introduse intr o comanda,daca nu exista o cantitate suficienta din acel produs in inregistrarea asociata produsului in baza de date atunci este afisata o eroare sau daca cantitatea nu corespunde formatului unui numar natural nenul este afisata o eroare.

Clasa QuantityValidator se ocupa cu validarea cantitatii unui produs introdus in baza de date,daca cantitatea nu corespunde formatului unui numar natural nenul este afisata o eroare.





Clasele BLL se ocupa cu apelarea metodelor continute de clasele DAO,sunt practic un mediator intre controller si clasele ce acceseaza baza de date,acestea actioneaza ca un mediator si filtru in acelasi timp deoarece apeleaza metodele de validare inainte de a apela metodele cerute de utilizator prin apasarea butoanelor,daca validatorii gasesc o neregula este afisata o eroare iar operatia nu este indeplinita.

Prima clasa de tip BLL este clasa ClientBLL,aceasta clasa are ca variabile instanta o lista de obiecte validator pentru a valida anumite aspecte ale obiectului client care se doreste inserat,are de asemenea un validator de ID independent si un obiect clientDAO pentru a apela metodele asociate bazei de date.

Constructorul acestei clase initializeaza variabile instanta,introduce in lista de validatori validatorii care corespund cu structura unei inregistrari de tip Client,cum ar fi validator de varsta validator de nume etc. , de asemenea este instantiat si obiectul clientDAO.

Aceasta clasa are 5 metode,prima dintre ele este metoda InsertClient(Client) care primeste ca argument un Obiect de tip Client,aceasta metoda aplica validatorii care corespund formatului de tip Client pe acest obiect iar daca inregistreaza Client trece testele este apelata metoda clientDAO.insert(Client).

A doua metoda in aceasta clasa este metoda DeleteByID(id) care primeste ca parametru un intreg care reprezinta id-ul clientului ce se vrea sters din baza de date.Este aplicat IDValidator pe id-ul trimis ca parametru iar daca nu trece testul este afisata o eroare ,daca este trecut testul este apelata metoda clientDAO.delete(id).

A treia metoda este EditClient(Client) , metoda aceasta aplica validatorii corespunzatori pe obiectul Client iar daca sunt trecute testele metoda apeleaza de trei metoda edit din ClientDAO pentru a edita campurile editabile din clasa Client,se apeleaza de 3 ori metoda edit deoarece unei constrangeri prezente in clasa abstracta DAO s-a ales trimiterea unui singur parametru spre edit.

A patra metoda este printAll() , aceasta metoda returneaza un ArrayList cu toti clientii prezenti la momentu acela in baza de date.

Cea de-a doua clasa din package-ul BLL este OrderBLL,aceasta clasa implementeaza ca si ClientBLL 5 metode,prima dintre ele fiind insert(Order) care primeste ca argument un obiect de tip Order,aplica pe acesta validatorii,daca obiectul nu trece testul este afisata o eroare,daca sunt trecute testele atunci este apelata metoda orderDAO.insert(Order).

A doua metoda implementata de aceasta clasa este DeleteByID(int id),aceasta metoda primeste ca parametru un intregu pe care se aplica IDValidator,daca ID-ul nu trece testul este afisat un mesaj de eroare altfel este apelata metoda orderDAO.deleteByID(id).

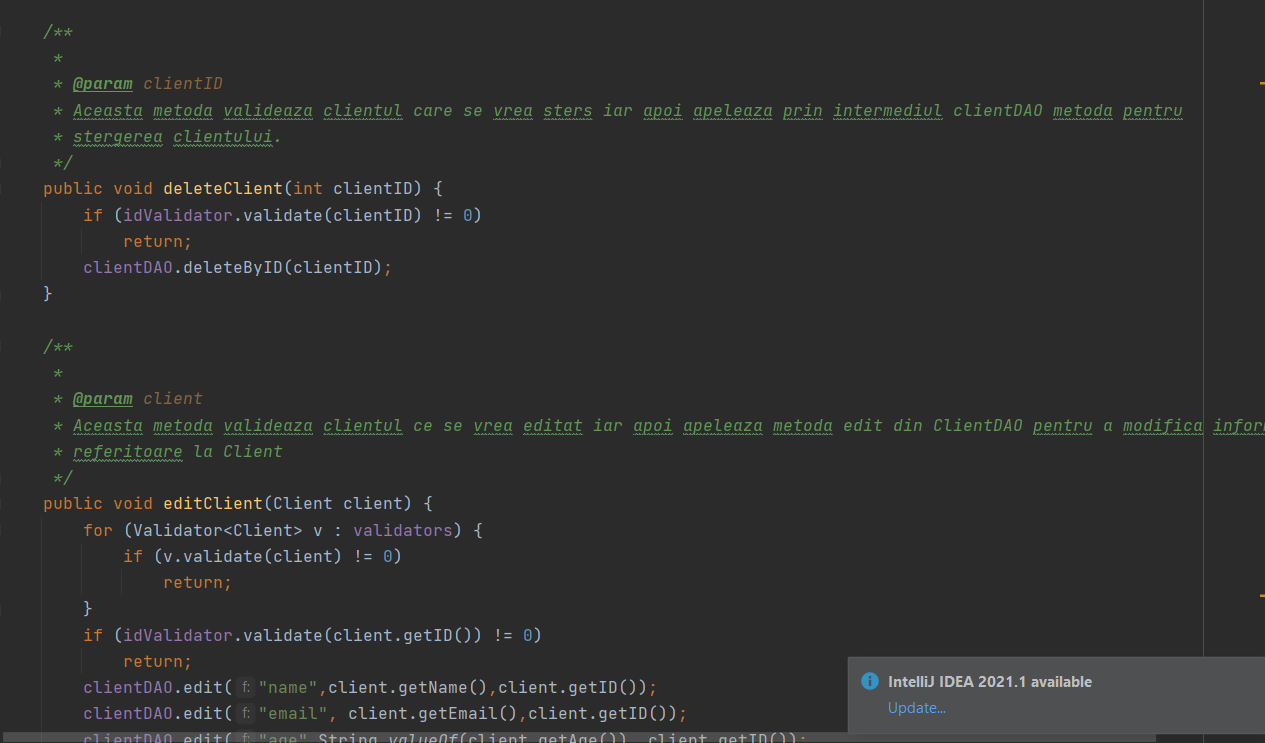
A treia metoda implementata de aceasta clasa este FindByID(int id),aceasta metoda aplica IDValidator pe intregul id si daca trece testul apeleaza orderDAO.findByID(id) care returneaza un obiect de tip Orders a carui id corespunde cu id ul primit ca parametru.

A patra metoda este EditOrder(Order),aplica pe obiectul Order validatorii corespunzatori dupa care apeleaza metoda orderDAO.edit(Order) pentru toate campurile editabile din Order pentru a schimba informatiile vechi cu informatiile noi introduse de utilizator.

A cincea metoda este printAll() care returneaza un ArrayList de Orders in urma apelarii metodeli orderDAO.showAll().

A treia si ultima clasa de tip BLL este ProductBLL,aceasta clasa implementeaza de asemenea aceleasi metode ca cele de mai sus,metoda de inserare produs care in urma validarii caracteristicilor produsului apeleaza inserarea aflata in productDAO,editarea unui produs prin metoda editProduct,stergerea unui produs prin metoda deleteByID(id) si afisarea tuturor produselor aflate la acel moment in baza de date prin metoda showAll().

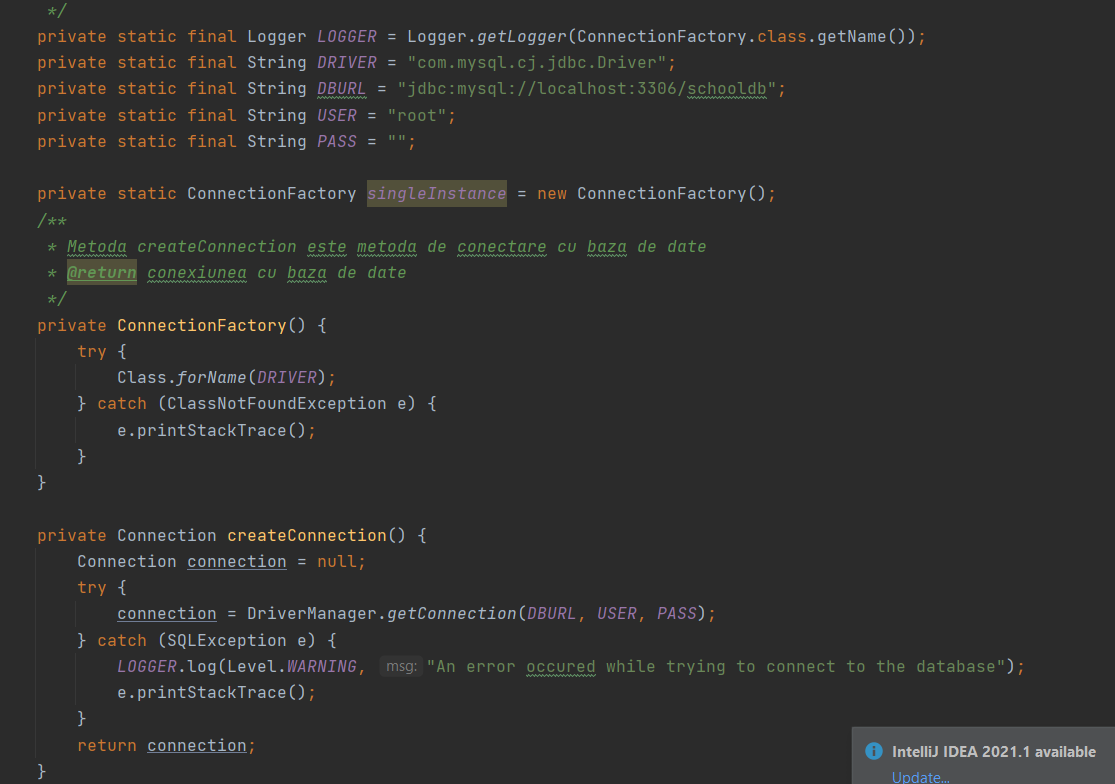
Clasele clientDAO,productDAO si orderDAO sunt clase care mostenesc clasa abstracta DAO care are metode generalizate si poate fi mostenita de diverse inregistrari cu diverse caracteristici,aceasta practica este una extrem de folostioare in cazul bazelor de date relationale cu foarte multe tabele.



Clasa ConnectionFactory se ocupa de conectarea aplicatiei la baza de date MySql ,aceasta clasa are 5 variabile instanta,LOGGER care este un obiect prestabilit Java care denumeste cumva cheia de logare ca fiind acea clasa,DRIVER,DBURL in care introducem adresa bazei de date la care vrem sa ne conectam,USER in care introducem numele de user al bazei de date si PASS unde se introduce parola bazei de date.

Clasa are o abordare de tip singletone,adica constructorul clasei este private,si o instanta a acestei clase este creata printr o metoda care verifica daca mai exista o alta instanta,prin acest mod se asigura unicitatea conexiunii la baza de date si cumva se evita operarea concurenta pe baza de date ce poate duce la inconsistente.

Clasa are metoda createConnection care creaza o conexiune cu baza de date ,datele bazei de date sunt date prin variabilele instanta precizate mai sus,o metoda getConnection care apeleaza metoda private createConnection iar daca nu exista nicio conexiune creaza una altfel o returneaza pe cea existenta si trei metode de close pentru statement,pentru resultSet si pentru connection.



Clasa AbstractDAO este clasa care implementeaza toate metodele generale de care este nevoie pentru baza de date,aceste metode sunt implementate intr-un stil generic astfel incat orice obiect ar apela acele metode sa fie viabila implementarea lor si sa functioneze corect.

In constructor clasa AbstractDAO genereaza tipul obiectului care extinde clasa,acel generic type.

Clasa are metode pentru generarea interogarilor care vor fi aplicate in metodele apelate de clasele BLL,metode cum ar fi createSelectQuery, createInsertQuery, createDeleteQuery, createEditQuery, createShowAllQuery.

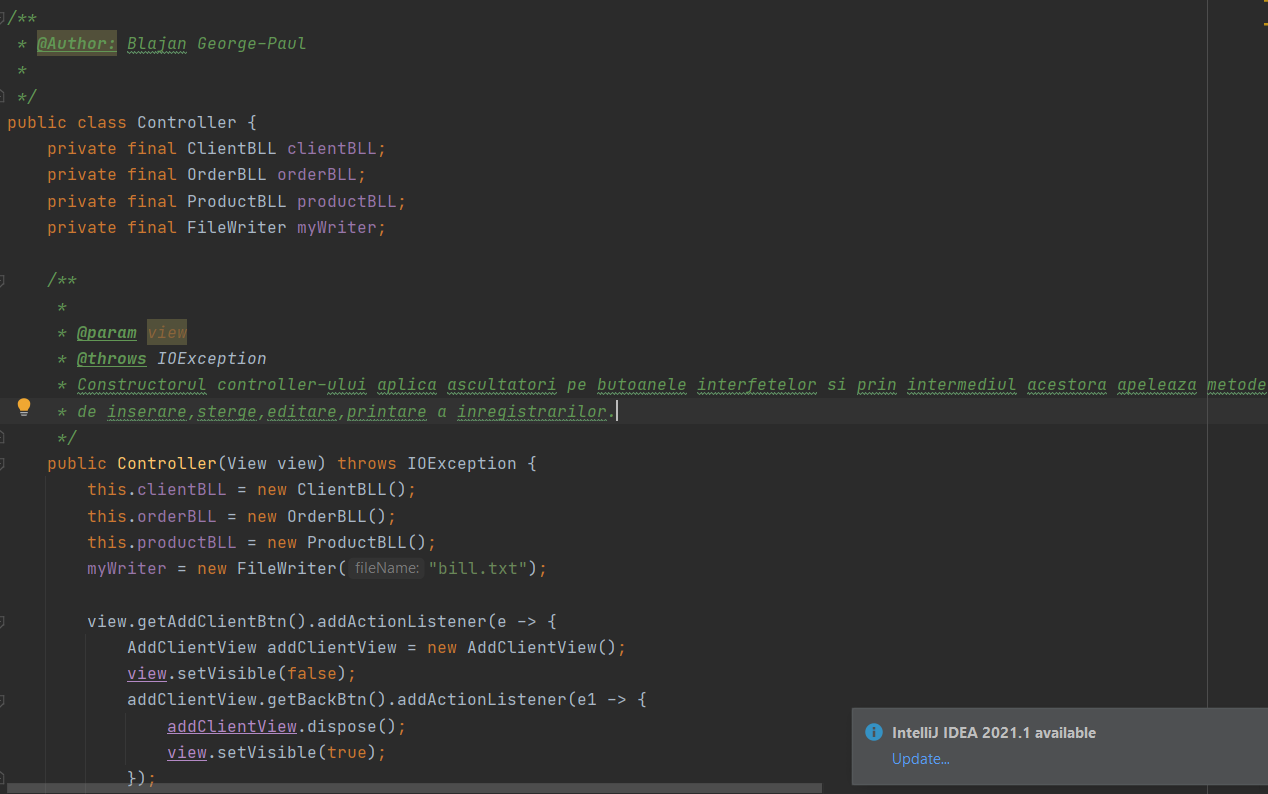
Clasa are metode care opereaza cu baza de date,si aplica interogarile formate anterior pentru a extrage rezultatul dorit in ResultSet.

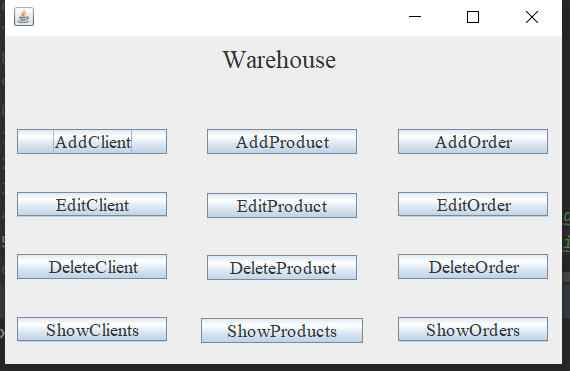
O astfel de metoda trebuie sa: creeze conexiunea cu baza de date,sa genereze obiectul de tip Statement cu interogarea construita anterior,sa initializeze Result Set,sa apeleze interogarea,sa sustraga rezultatul in obiectul de tip ResultSet,sa inchida conexiunea cu baza de date,cu statementul si cu resultSet.

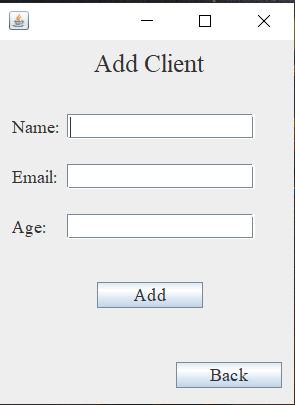
Clasele din Model,Client,Product si Orders sunt maparea tabelelor din baza de date relationala,aceste clase contin exact aceleasi campuri ca si cele din baza de date si au metode de get si de set in principal,nicio alta metoda nu este implementata aici cu exceptia acestora si a toString-ului.

Clasa controller este clasa care orchestreaza aceasta aplicatie,aceasta clasa primeste clasa View care este prima interfata GUI,unde sunt toate butoanele care reprezinta comenzile,si construieste ascultatori pentru aceste butoane,la apasarea unui buton care reprezinta o comanda controllerul instantiaza un view de acel tip si aplica pe acel view alti ascultatori pentru butoanele de operatie si de back.

In acest proiect avem 13 clase de View,clasa principala care contine butoanele de operatii si clasele secundare care reprezinta interfetele unde utilizatorul introduce date in baza de date.







# ***Rezultate:***

Acest proiect a fost testat de mana pentru a verifica functionalitatea acestuia in toate cazurile posibile.

# ***Concluzii:***

Ca extindere acest proiect ar putea fi perfectionat si sa serveasca ca un adevarat Scheduler pentru comenzile plasate in cadrul unui depozit de produse.

# ***Bibliografie:***

[O introducere la API-ul Java Reflection. Diferența de comportament (interweber.ru)](https://interweber.ru/ro/application/vvedenie-v-java-reflection-api-raznica-v-povedenii-funkcii-programmy-rezervnogo.html)