

Gestionarea comenziilor

Tema 3



Popesc Ariadna-Ioana

Grupa 30228

Cuprins

[**1.** **Obiectivul temei** 2](#_Toc136273326)

[**2.** **Analiza problemei** 2](#_Toc136273327)

[**3.** **Analiza problemei** 4](#_Toc136273328)

[**4.** **Implementare** 5](#_Toc136273329)

[**5. Concluzii** 20](#_Toc136273330)

[**6.** **Bibliografie** 21](#_Toc136273331)

# **Obiectivul temei**

*Obiectivul principal*

Scopul principal al temei „ Management-ul comenzilor” constă în proiectarea și implementarea unei aplicații software pentru a gestiona comenzile clienților pentru un depozit.

*Obiective secundare*

* Analizarea problemei și identificarea cerințelor
* Proiectarea aplicației de gestionare
* Implementarea aplicației de gestionare a comenzilor
* Testarea comenzilor aplicației de gestionare

# **Analiza problemei**

Această temă are ca obiectiv propunerea, proiectarea și implementarea unui sistem de gestionare a comenzilor dintr-un depozit.

Cerințele non-funcționale sunt:

* Simularea aplicației ar trebui să fie intuitivă și ușor de folosit de către utilizator

Cerințele funcționale sunt:

* Simularea aplicației ar trebui să permită angajatului să introducă un nou client
* Simularea aplicației ar trebui să permită angajatului să introducă un nou produs
* Simularea aplicației ar trebui să permită angajatului să introducă o nouă comandă
* Caz de utilizare: adăugare de produs
* Actor principal: angajat
* Scenariul principal de succes:
* Angajatul selectează opțiunea de a adăuga un nou produs
* Aplicația va afișa pe ecran o formă în care detaliile produsului ar trebui să fie inserate
* Angajatul introduce numele produsului, prețul și stocul curent
* Angajatul apasă butonul de “Add”
* Apoi aplicația reține datele produselor în baza de date și afișează un mesaj de înștiințare
* Secvență alternativă: date invalide pentru datele produsului
* Utilizatorul introduce valori negative pentru stock-ul produselor
* Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă valori valide-Scenariul revine la pasul 3

**Diagrama Use-Case pentru aplicația “Procesarea cozilor”**

O imagine care conține diagramă, cerc, Color

Descriere generată automat

# **Analiza problemei**

* Arhitectura conceptuală

O imagine care conține text, captură de ecran, diagramă, Font

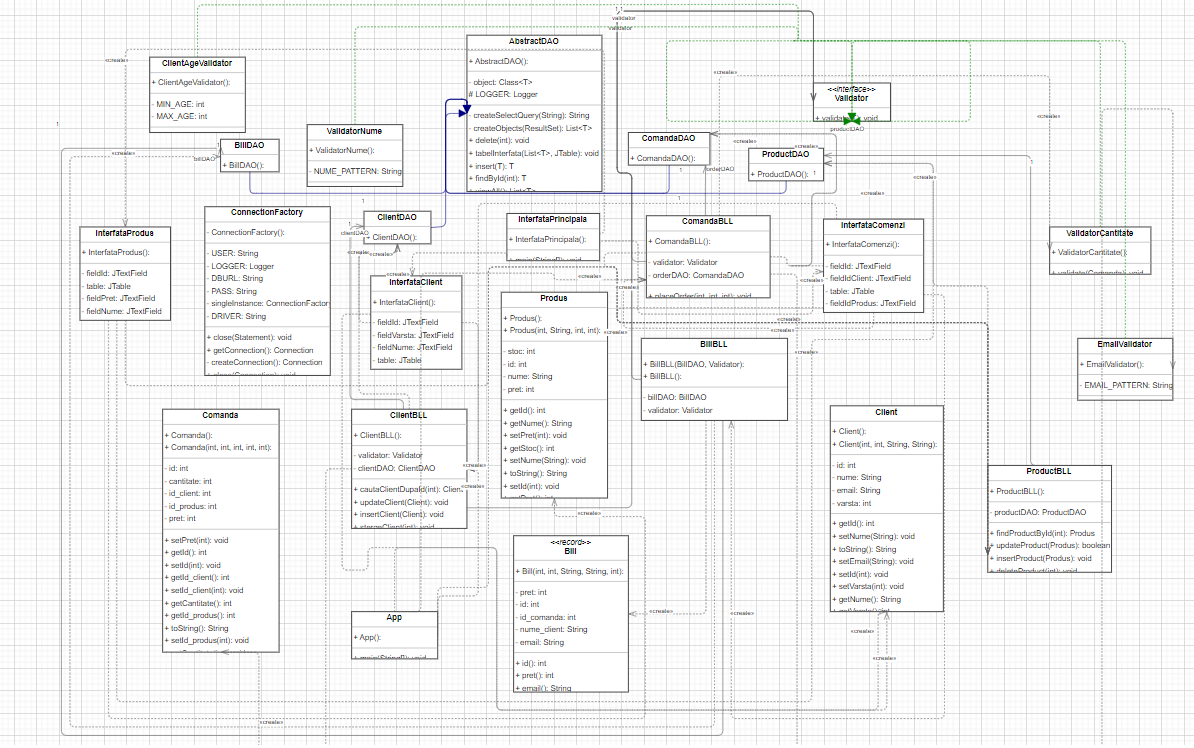
Descriere generată automat

* Diagrama de pachete

O imagine care conține diagramă, linie, text, Desen tehnic

Descriere generată automat

* Diagrama UML de clase



# **Implementare**

Proiectul este împărțit în 5 pachete de clase. Pachetele sunt:

1. **Model**, în care se regăsesc următoarele clase:
2. **Client:**

* În care există un id, vârstă, nume și email pentru fiecare Client în parte. Sunt generate setterele și getterele pentru fiecare obiect în parte, precum și contructorul si metoda toString.
* Plus un constructor fără parametrii.
* public Client(int id, int varsta, String nume, String email) {  
   this.id = id;  
   this.varsta = varsta;  
   this.nume = nume;  
   this.email = email;  
  }  
  public Client(){  
    
  }  
    
  @java.lang.Override  
  public java.lang.String toString() {  
   return "Client{" +  
   "id=" + id +  
   ", varsta=" + varsta +  
   ", nume='" + nume + '\'' +  
   ", email='" + email + '\'' +  
   '}';  
  }

1. **Produs:**

- În care există un id, nume, preț și stoc pentru fiecare Produs în parte. Sunt generate setterele și getterele pentru fiecare obiect în parte, precum și contructorul si metoda toString.

- Plus un constructor fără parametrii.

public Produs(int id, String nume, int pret, int stoc) {  
 this.id = id;  
 this.nume = nume;  
 this.pret = pret;  
 this.stoc = stoc;  
}  
  
public Produs() {  
  
}  
  
@java.lang.Override  
public java.lang.String toString() {  
 return "Product{" +  
 "id =" + id +  
 ", nume\_produs=" + nume +  
 ", pret=" + pret +  
 ", stoc=" + stoc +  
 '}';  
}

1. **Comanda**

-În care există un id, nume, preț și stoc pentru fiecare Produs în parte. Sunt generate setterele și getterele pentru fiecare obiect în parte, precum și contructorul si metoda toString.

- Plus un constructor fără parametrii.

1. **Bill**

* Reprezintă o factură, fiind o clasă realizată cu JavaRecors.

1. public record Bill(int id, int id\_comanda, String nume\_client, String email, int pret) {  
     
   }
2. **Business\_Logic** în care se găsesc următoarele clase, care implementează logica aplicației.
3. **ClientBLL**

* În clasa ClientBLL se apelează clientDAO și sunt implementate metodele: cautaClientDupaId care va cauta un client in functie de id, insertClient care va insera câte un client, findAllClients care va fi folosit pentru a afișa clienții, updateClient care va adapta fiecare client și stergeClient care va șterge clientul în funcție de id.
* Înaintea fiecărei metode au fost explicate funcționările acestora, folosind comentarii JavaDoc.
* */\*\*  
   \* metoda insereaza un client si valideaza datele introduse  
   \** ***@param*** *cl  
   \* nu returneaza nimic  
   \*/*public void insertClient(Client cl){  
   validator.validate(cl);  
   clientDAO.insert(cl);  
  }

1. **ComandaBLL**

* În clasa ComandaBLL se va apela ComandaDAO, iar metoda placeOrder implementată plasează o comandă în sistem, actualizănd stocul produsului, ștergând în cazul în care stocul e 0 și metoda findAllOrders adaugă toate comenziile într-o listă pentru a fi afișată mai târziu.
* */\*\*  
   \* aceasta metoda plaseaza o comanda in sistem, actualizand stocul produsului, stergand in cazul in care stocul e 0  
   \** ***@param*** *idClient  
   \** ***@param*** *idProdus  
   \** ***@param*** *cantitate  
   \** ***@throws*** *Exception  
   \*/*public void placeOrder(int idClient, int idProdus, int cantitate) throws Exception {  
   ProductBLL productBLL = new ProductBLL();  
   Produs produs = productBLL.findProductById(idProdus);  
   Comanda comand = new Comanda(0, idClient, idProdus, cantitate, produs.getPret() \* cantitate);  
   validator.validate(comand);  
   orderDAO.insert(comand);  
   productBLL.updateProduct(produs);  
   produs = productBLL.findProductById(idProdus);  
   if(produs.getStoc() == 0){  
   productBLL.deleteProduct(produs.getId());  
   }  
  }

1. **ProductBLL**

* În clasa ProductBLL se va apela productDAO, iar metodele vor căuta produsul după id, inserează produsul,, îl actualizează și îl șterge.
* */\*\*  
   \* adauga toate produsele intr-o lista  
   \* nu are parametrii  
   \** ***@return*** *lista de produse  
   \*/*public List<Produs> findAllProducts() {  
   List<Produs> list = productDAO.viewAll();  
   if (list == null) {  
   throw new NoSuchElementException("NU exista produse!!");  
   }  
   return list;  
  }  
  */\*\*  
   \* verifica daca produsul a putut fi actualizat sau nu  
   \** ***@param*** *pd  
   \** ***@return*** *\** ***@throws*** *Exception  
   \*/*public boolean updateProduct(Produs pd) throws Exception {  
   if (productDAO.update(pd) == null) {  
   throw new Exception("Produsul nu a putut fi actualizat!");  
   }  
   return true;  
  }  
    
  */\*\*  
   \* sterge produsul in functie de id  
   \** ***@param*** *id  
   \* nu returneaza nimic  
   \*/*public void deleteProduct(int id) {  
   Produs p = productDAO.findById(id);  
   productDAO.delete(p.getId());  
  }

**d. BillBLL**

- Clasa BillBLL face legătura dintre datele din table și interfață.

private BillDAO billDAO = new BillDAO();  
private Validator validator;  
  
public BillBLL(BillDAO billDAO, Validator validator) {  
 this.billDAO = billDAO;  
 this.validator = validator;  
}  
  
public BillBLL(){  
  
}  
  
public void billInsert(int id, int id\_comanda, String nume\_client, String email, int pret) throws Exception {  
 Bill factura = new Bill(id, id\_comanda, nume\_client, email, pret);  
 billDAO.insert(factura);  
  
}

1. **Connection** care include o singură clasă:

Clasa **ConnectionFactory**, care are rolul de a lega baza de date din MySQL cu aplicația implementată.

public class ConnectionFactory {  
  
 private static final Logger *LOGGER* = Logger.*getLogger*(ConnectionFactory.class.getName());  
 private static final String *DRIVER* = "com.mysql.cj.jdbc.Driver";  
 private static final String *DBURL* = "jdbc:mysql://localhost:3306/comenzi";  
 private static final String *USER* = "root";  
 private static final String *PASS* = "Sql\_LabTp2@3";  
  
 private static ConnectionFactory *singleInstance* = new ConnectionFactory();  
  
 private ConnectionFactory() {  
 try {  
 Class.*forName*(*DRIVER*);  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private Connection createConnection() {  
 Connection connection = null;  
 try {  
 connection = DriverManager.*getConnection*(*DBURL*, *USER*, *PASS*);  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, "An error occured while trying to connect to the database");  
 e.printStackTrace();  
 }  
 return connection;  
 }  
  
 public static Connection getConnection() {  
 return *singleInstance*.createConnection();  
 }  
  
 public static void close(Connection connection) {  
 if (connection != null) {  
 try {  
 connection.close();  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, "An error occured while trying to close the connection");  
 }  
 }  
 }  
  
 public static void close(Statement statement) {  
 if (statement != null) {  
 try {  
 statement.close();  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, "An error occured while trying to close the statement");  
 }  
 }  
 }  
  
 public static void close(ResultSet resultSet) {  
 if (resultSet != null) {  
 try {  
 resultSet.close();  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, "An error occured while trying to close the ResultSet");  
 }  
 }  
 }  
}

1. **Data\_Access** include 5 clase, acestea fiind:
   1. **AbstractDAO**

* Această clasă conține metodele care fac interogările pe baza de date.
* public class AbstractDAO<T> {  
    
    
   protected static final Logger *LOGGER* = Logger.*getLogger*(AbstractDAO.class.getName());  
   private final Class<T> object; // declar un obiect de tipul clasa  
    
   */\*\*  
   \* in constructor salvam in object clasa generica specificata in implementarea subclaselor AbstractDao  
   \* adica ClientDao, ComandaDAO si ProdudtDAO  
   \*/* public AbstractDAO() {  
   this.object = (Class<T>) ((ParameterizedType) getClass().getGenericSuperclass()).getActualTypeArguments()[0];  
   }  
    
   */\*\*  
   \* selecteaza o inregistrare din baza de date asociata cu id-ul specificat, creeaza o instanța a obiectului T si seteaza valorile campurilor obiectului folosind valorile din baza de date.  
   \*  
   \** ***@param*** *//id  
   \** ***@return*** *un obiect de tip T asociat cu acel id din baza de date  
   \*/* private String createSelectQuery(String field) {  
   StringBuilder sb = new StringBuilder();  
   sb.append("SELECT ");  
   sb.append(" \* ");  
   sb.append(" FROM ");  
   sb.append(object.getSimpleName());  
   sb.append(" WHERE " + field + " =?");  
   return sb.toString();  
   }  
    
   */\*\*  
   \* metoda cauta obiectul cu id ul dat ca parametru  
   \** ***@param*** *id  
   \** ***@return*** *\*/* public T findById(int id) {  
   Connection connection = null;  
   PreparedStatement statement = null;  
   ResultSet resultSet = null;  
   String query = createSelectQuery("id");  
   try {  
   connection = ConnectionFactory.*getConnection*();  
   statement = connection.prepareStatement(query);  
   statement.setInt(1, id);  
   resultSet = statement.executeQuery();  
    
   return createObjects(resultSet).get(0);  
   } catch (SQLException e) {  
   *LOGGER*.log(Level.*WARNING*, object.getName() + "DAO:findById " + e.getMessage());  
   } finally {  
   ConnectionFactory.*close*(resultSet);  
   ConnectionFactory.*close*(statement);  
   ConnectionFactory.*close*(connection);  
   }  
   return null;  
   }  
    
   */\*\*  
   \* metoda creeaza obiectele si le pune intr o lista  
   \** ***@param*** *resultSet  
   \** ***@return*** *\*/* private List<T> createObjects(ResultSet resultSet) {  
   List<T> list = new ArrayList<T>();  
   Constructor[] ctors = object.getDeclaredConstructors();  
   Constructor ctor = null;  
   for (int i = 0; i < ctors.length; i++) {  
   ctor = ctors[i];  
   if (ctor.getGenericParameterTypes().length == 0)  
   break;  
   }  
   try {  
   while (resultSet.next()) {  
   ctor.setAccessible(true);  
   T instance = (T)ctor.newInstance();  
   for (Field field : object.getDeclaredFields()) {  
   String fieldName = field.getName();  
   Object value = resultSet.getObject(fieldName);  
   PropertyDescriptor propertyDescriptor = new PropertyDescriptor(fieldName, object);  
   Method method = propertyDescriptor.getWriteMethod();  
   method.invoke(instance, value);  
   }  
   list.add(instance);  
   }  
   } catch (InstantiationException e) {  
   e.printStackTrace();  
   } catch (IllegalAccessException e) {  
   e.printStackTrace();  
   } catch (SecurityException e) {  
   e.printStackTrace();  
   } catch (IllegalArgumentException e) {  
   e.printStackTrace();  
   } catch (InvocationTargetException e) {  
   e.printStackTrace();  
   } catch (SQLException e) {  
   e.printStackTrace();  
   } catch (IntrospectionException e) {  
   e.printStackTrace();  
   }  
   return list;  
   }  
    
   */\*\*  
   \** ***@return*** *o lista care va contine toate inregistrarile dintr-o tabela asociate clasei de tip T din baza de date  
   \*/* public List<T> viewAll() {  
   StringBuilder sb = new StringBuilder();  
   sb.append("SELECT \* FROM ");  
   sb.append(object.getSimpleName());  
    
   Connection connection = null;  
   PreparedStatement statement = null;  
   ResultSet resultSet = null;  
   String query = sb.toString();  
   try {  
   connection = ConnectionFactory.*getConnection*();  
   statement = connection.prepareStatement(query);  
   resultSet = statement.executeQuery();  
   return createObjects(resultSet);  
   } catch (SQLException e) {  
   *LOGGER*.log(Level.*WARNING*, object.getName() + "DAO:findById " + e.getMessage());  
   } catch (IndexOutOfBoundsException e) {  
   *LOGGER*.log(Level.*WARNING*, object.getName() + "DAO:findById " + e.getMessage());  
   } finally {  
   ConnectionFactory.*close*(resultSet);  
   ConnectionFactory.*close*(statement);  
   ConnectionFactory.*close*(connection);  
   }  
   return null;  
   }  
    
    
   */\*\*  
   \* functia insert va insera un obiect de tipul T  
   \** ***@param*** *obj\_insert  
   \** ***@return*** *obiectul inserat  
   \*/* public T insert(T obj\_insert) {  
   // construieste query-ul pentru insert, pentru fiecare tabel in parte  
   StringBuilder sb = new StringBuilder();  
   sb.append("INSERT INTO ");  
   sb.append(object.getSimpleName());  
   int currentParameter = 1;  
   for (Field f: obj\_insert.getClass().getDeclaredFields()) {  
   if (f.getName().equals("id") == false) // se sare peste id pt ca el are autoincrement in tabela  
   {  
   if (currentParameter == 1) { // daca i primu element, trebuie sa deschida parantezele  
   sb.append("(" + f.getName()); // pune numele field-ului/coloanei  
   } else {  
   sb.append(", " + f.getName());  
   }  
   currentParameter++;  
   }  
   }  
   sb.append(") VALUES ");  
   for (int i = 1; i <= obj\_insert.getClass().getDeclaredFields().length - 1; i++) {  
   if (i == 1) { // daca i primu element, va trebui sa deschida parantezele de la values  
   sb.append("(?"); // salvam un loc pentru a introduce valoarea ulterioara a parametrului din tabel  
   }  
   else {  
   sb.append(", ?"); // salvam un loc pentru a introduce valoarea ulterioara a parametrului din tabel } }  
   sb.append(")");  
   currentParameter = 1;  
   Connection connection = null;  
   PreparedStatement statement = null;  
   String query = sb.toString();  
    
   try {  
   connection = ConnectionFactory.*getConnection*();  
   statement = connection.prepareStatement(query);  
   for (Field f: obj\_insert.getClass().getDeclaredFields()) {  
   f.setAccessible(true);  
   if (f.getName().equals("id")) // sare peste id pentru ca nu s-a precizar mai sus, inserandu-se doar celelalte valori  
   continue; // id are autoincrement si nu trebuie inserat manual  
   if (f.getType().getSimpleName().equals("String")) {  
   statement.setString(currentParameter, (String)f.get(obj\_insert));  
   } else if (f.getType().getSimpleName().equals("int")) {  
   statement.setInt(currentParameter, (int)f.get(obj\_insert));  
   }  
   currentParameter += 1;  
   }  
   statement.execute();  
   return obj\_insert;  
   } catch (SQLException e) {  
   *LOGGER*.log(Level.*WARNING*, object.getName() + "DAO:insert " + e.getMessage());  
   } catch (IllegalAccessException e) {  
   *LOGGER*.log(Level.*WARNING*, object.getName() + "DAO:insert " + e.getMessage());  
   } finally {  
   ConnectionFactory.*close*(statement);  
   ConnectionFactory.*close*(connection);  
   }  
   return null;  
   }  
    
   */\*\*  
   \* Metoda actualizeaza campurile fiecarui tabel  
   \** ***@param*** *obiect  
   \** ***@return*** *\*/* public T update(T obiect) {  
   // construieste query-ul pentru update, pentru fiecare tabel in parte  
   StringBuilder str = new StringBuilder();  
   str.append("UPDATE ");  
   str.append(object.getSimpleName());  
   str.append(" SET ");  
   int currentParameter = 1;  
   for (Field f: obiect.getClass().getDeclaredFields()) {  
   if (!f.getName().equals("id")) // daca coloana nu e id  
   {  
   if (currentParameter != 1) {  
   str.append(", ");  
   }  
   str.append(f.getName()); // pune numele field-ului/coloanei  
   str.append(" = ?"); // salvam un loc pentru a introduce valoarea ulterioara a parametrului din tabel  
   currentParameter++;  
   }  
   }  
   str.append(" WHERE id = ?"); //update tabela set(nume=?,varsta=?) where id=?  
    
   Connection connection = null;  
   PreparedStatement statement = null;  
   String query = str.toString();  
    
   int idForUpdate = -1;  
   try {  
   connection = ConnectionFactory.*getConnection*();  
   statement = connection.prepareStatement(query);  
   currentParameter = 1;  
   for (Field f: obiect.getClass().getDeclaredFields()) {  
   f.setAccessible(true); // pentru a accesa elementele private dintr-o clasa  
   if (f.getName().equals("id")) {  
   idForUpdate = (int)f.get(obiect);  
   continue;  
   }  
   if (f.getType().getSimpleName().equals("String")) {  
   statement.setString(currentParameter, (String)f.get(obiect));  
   } else if (f.getType().getSimpleName().equals("int")) {  
   statement.setInt(currentParameter, (int)f.get(obiect));  
   }  
   currentParameter += 1;  
   }  
   statement.setInt(currentParameter, idForUpdate); // adaug id-ul la final pentru coloana respectiva  
   statement.execute();  
   return obiect;  
   } catch (SQLException e) {  
   *LOGGER*.log(Level.*WARNING*, object.getName() + "DAO:update " + e.getMessage());  
   } catch (IllegalAccessException e) {  
   *LOGGER*.log(Level.*WARNING*, object.getName() + "DAO:update " + e.getMessage());  
   } finally {  
   ConnectionFactory.*close*(statement);  
   ConnectionFactory.*close*(connection);  
   }  
   return null;  
   }  
    
   */\*\*  
   \* Metoda sterge o inregistrare din tabela corespunzatoare obiectului T  
   \** ***@param*** *id ul inregistrarii care se doreste sa se stearga  
   \*/* public void delete(int id) {  
    
   StringBuilder str = new StringBuilder();  
   str.append("DELETE FROM ");  
   str.append(object.getSimpleName());  
   str.append(" WHERE id =?");  
    
   Connection connection = null;  
   PreparedStatement statement = null;  
   String query = str.toString();  
    
   try {  
   connection = ConnectionFactory.*getConnection*();  
   statement = connection.prepareStatement(query);  
   statement.setInt(1, id);  
   statement.execute();  
   } catch (IndexOutOfBoundsException e) {  
   *LOGGER*.log(Level.*WARNING*, object.getName() + "DAO:deleteById " + e.getMessage());  
   } catch (SQLException e) {  
   *LOGGER*.log(Level.*WARNING*, object.getName() + "DAO:deleteById " + e.getMessage());  
   } finally {  
   ConnectionFactory.*close*(statement);  
   ConnectionFactory.*close*(connection);}  
   }  
    
   */\*\*  
   \*  
   \** ***@param*** *list  
   \** ***@param*** *tabel  
   \** ***@throws*** *IllegalAccessException  
   \*/* public void tabelInterfata(List<T> list, JTable tabel) throws IllegalAccessException {  
   if (!list.isEmpty()) {  
    
   int nrCol = 0;  
   for (Field field : list.get(0).getClass().getDeclaredFields()) {  
   nrCol++;  
   }  
   String[] coloana = new String[nrCol];  
   int i = 0;  
   for (Field field : list.get(0).getClass().getDeclaredFields()) {  
   coloana[i] = field.getName();  
   i++;  
   }  
    
   String[][] data = new String[list.size()][nrCol];  
   int j = 0;  
   for(i = 0; i < list.size(); i++) // parcurg lista de obiecte  
   {  
   j = 0;  
   for (Field field : list.get(i).getClass().getDeclaredFields()) { // parcurg fiecare camp in parte  
   field.setAccessible(true);  
   try {  
   data[i][j] = field.get(list.get(i)) + "";  
   } catch (IllegalArgumentException | IllegalAccessException e) {  
   e.printStackTrace();  
   }  
   j++;  
   }  
   }  
   DefaultTableModel model = new DefaultTableModel(data, coloana);  
   tabel.setModel(model);  
   }  
   }
* }
  1. **BillDAO**
  2. **ClientDAO**
  3. **ComandaDAO**
  4. **ProductDAO**
* Toate aceste 4 clase extend clasa AbstractDAO, unde se regăsesc toate implementările.

1. **Presentation** care include cele 4 interfețe, și anume:
   1. O imagine care conține text, captură de ecran, diagramă, număr

      Descriere generată automatInterfataClient
   2. O imagine care conține text, captură de ecran, diagramă, număr

      Descriere generată automatInterfataComenzi
   3. O imagine care conține text, captură de ecran, Font, Dreptunghi

      Descriere generată automatInterfataPrincipala
   4. O imagine care conține text, captură de ecran, diagramă, număr

      Descriere generată automatInterfataProdus

-Interfețele InterfataClient, InterfataComenzi și InterfataProdus sunt legate de interfața principală prin actionListener-urile alea fiecărui buton, specific cu fiecare nume al ferestrei în care se va folosi.

# **5. Concluzii**

Dezvoltarea acestei teme m-a ajutat sa invat despre modul de creare a metodelor folosind reflexii, utilizarea inregistrarilor si comunicarea cu o baza de date mysql folosind concepte POO.

În ceea ce priveste modalitatile de dezvoltare ulterioara a aplicatiei, acestea ar putea fi:

* Adaugarea de noi parametri pentru mai multe detalii legate de tabele.
* Crearea unor ferestre specifice unui angajat si a unui client.

# **Bibliografie**

[**https://stackoverflow.com/questions/66910411/regular-expression-to-validate-a-name**](https://stackoverflow.com/questions/66910411/regular-expression-to-validate-a-name)

[**https://dsrl.eu/courses/pt/**](https://dsrl.eu/courses/pt/)

[**https://stackoverflow.com/questions/1990817/how-to-make-a-jtable-non-editable**](https://stackoverflow.com/questions/1990817/how-to-make-a-jtable-non-editable)

[**https://app.diagrams.net/**](https://app.diagrams.net/)