Tema 3

Moldovan Radu Octavian
Grupa 30224

CUPRINS

1. Obiectivul temei 3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4
3. Proiectare 8
4. Implementare 11
5. Concluzii 16
6. Bibliografie 16

1. Obiectivul temei

Obiectivul principal:

Obiectivul principal presupune crearea unei aplicatii usor de utilizat, pentru gestionarea comenzilor clientilor dintr-un depozit.

Obiectivele secundare:

- Verificarea input-ului introdus pentru a asigura o legatură unu la unu, între ceea ce este introdus în casetele text, corespunzatoare parametrilor de intrare (insert, update, delete, order), și ceea ce se află in memoria aplicației;
- Modificarea clientilor doar prin selectarea acestora din tabelele puse la dispozitie.
- Interceptarea tuturor erorilor pentru a evita o situație în care programul se află într-o stare necunoscută;
- Informarea utilizatorului despre erori prin pop-up-uri si coduri de eroare care pot fi depanate mai departe consultând un manual specific pentru erori;
- Informarea utilizatorului cu privință la corectitudinea input-ului introdus de la tastatură prin pop-up-uri;

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

Ideea principala în realizarea aplicatie de gestionare este de a pune la dispozitie utilizatorului o interfata usor de utilizat, prin comenzi si o modalitati de operare sugestive. Interfata grafica dispune de ferestre specifice pentru fiecare operatie de inserare, stergere, modificare sau crearea unei comenzi, tabelele din baza de date fiind disponibile clientului ce utilizeaza aplicatia.

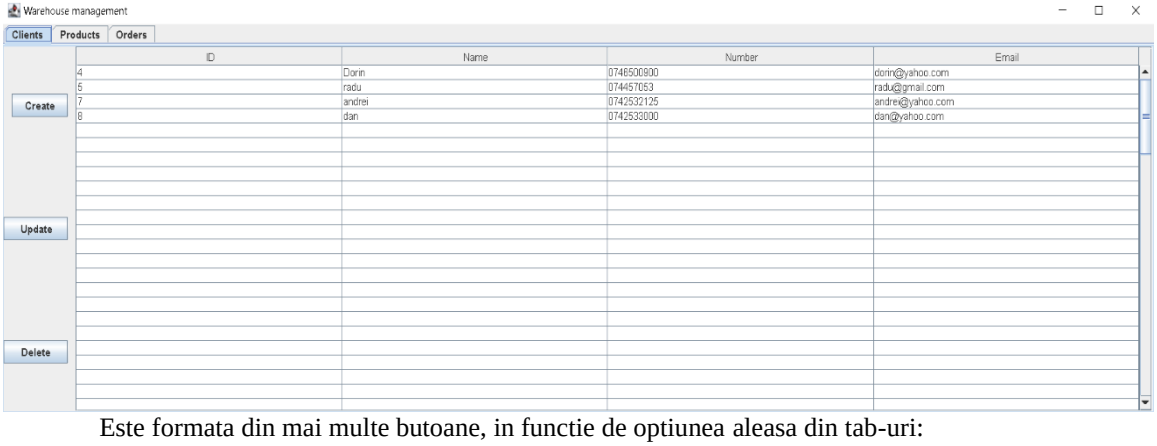
Scenarii posibile:

1. Introducerea datelor in mod eronat, care ar putea face realizarea producere unei exceptii la inserarea/stergere/modificarea datelor in/din tabel:
    - Introducerea de numere negative in casetele destinate pentru stock/amount;
    - Se activeaza un pop-up ce semnalează ca datele nu sunt introduse corect;
2. Introducerea partiala a unor date de input:
    - În acest caz, este activat un pop-up ce semnalează ca, casetele de text respective sunt goale.

Modelare:

Aplicația propriu-zisă este formată dintr-o fereastra principala si mai multe ferestre secundare, specifice fircarei operatii.

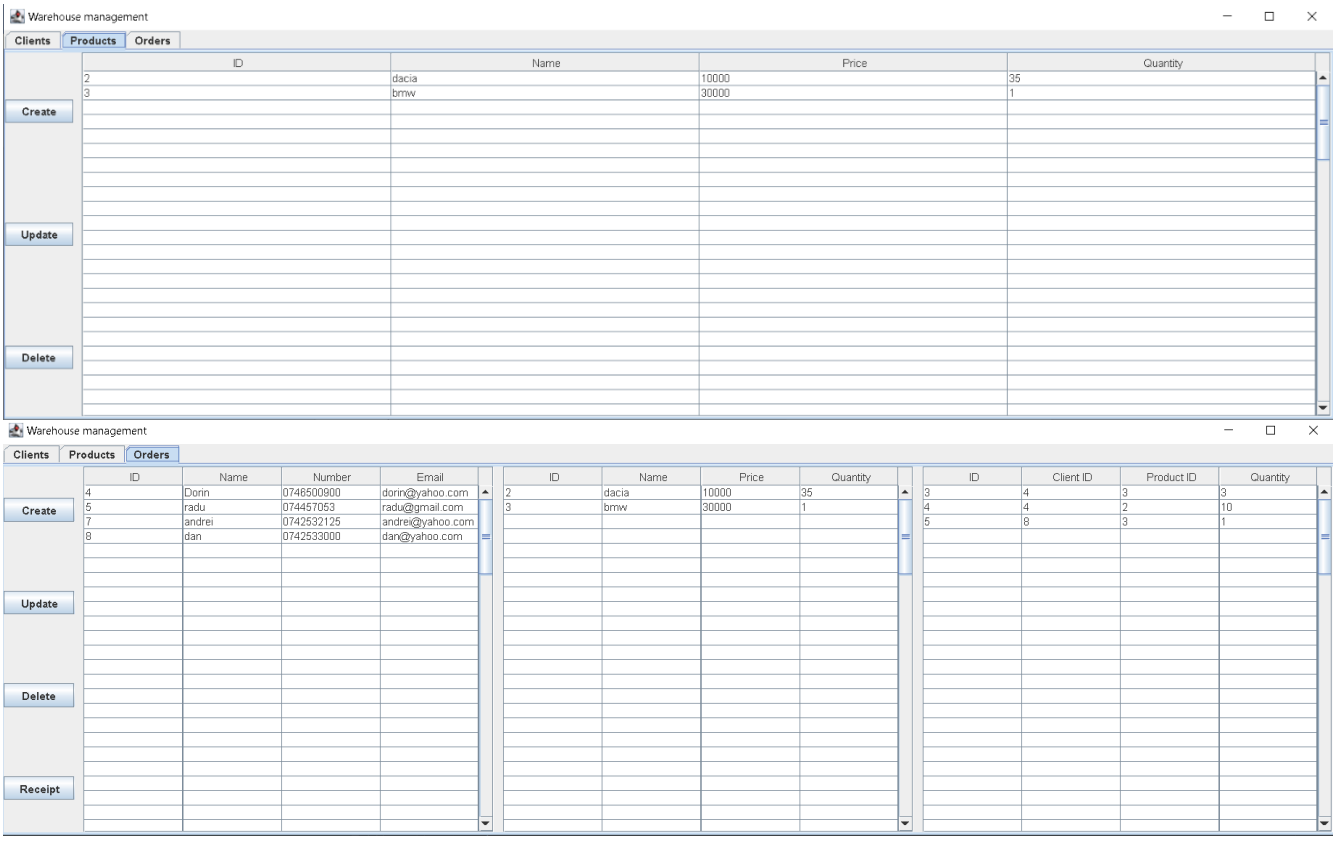
Fereastra principala:



Este formata din mai multe butoane, in functie de optiunea aleasa din tab-uri:

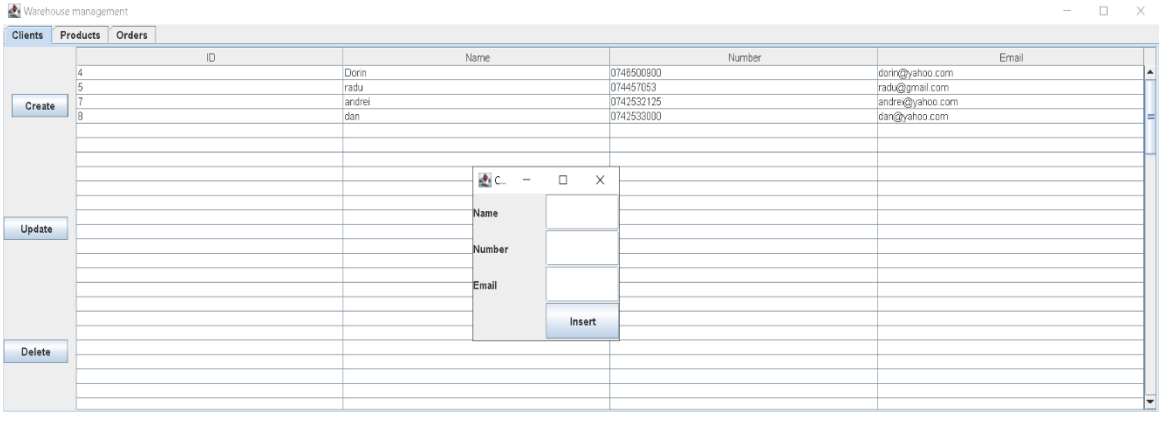
- Daca sunt selectate tabelele Clients si Products, exista 3 butoane:
  - Unul pentru operatia de inserare „Insert”;
  - Unul pentru operatia de modificare „Update”;
  - Unul pentru operatia de stergere „Delete”;
- Daca este selectat tabelul Orders:
  - Unul pentru operatia de modificare „Update”;
  - Unul pentru operatia de stergere „Delete”;
  - Unul pentru operatia de order „Order”;
  - Unul pentru a tiparii factura „Receipt”.

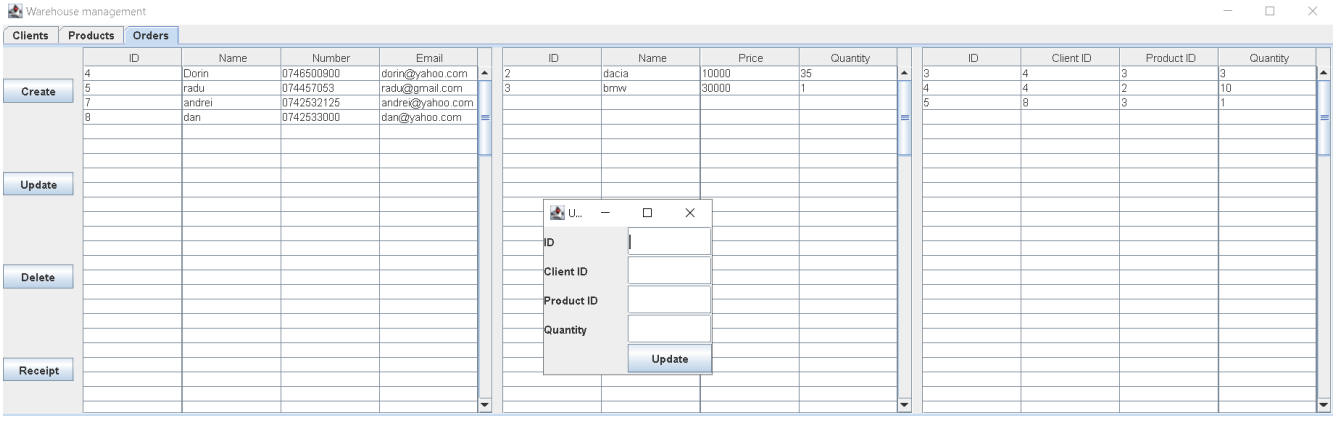
In cazul in care sunt selectate optiunile de Client si Product atunci sunt disponibile operatiile doar pe tabelele respective iar in cazul in care este selectata operatia de Ordert, sunt disponibile toate cele 3 tabele, dar se pot realiza operatii de modificare si stergere doar pe tabela de Order, in timp de operatia de „Order” este realizata pe toate cele 3 tabele.

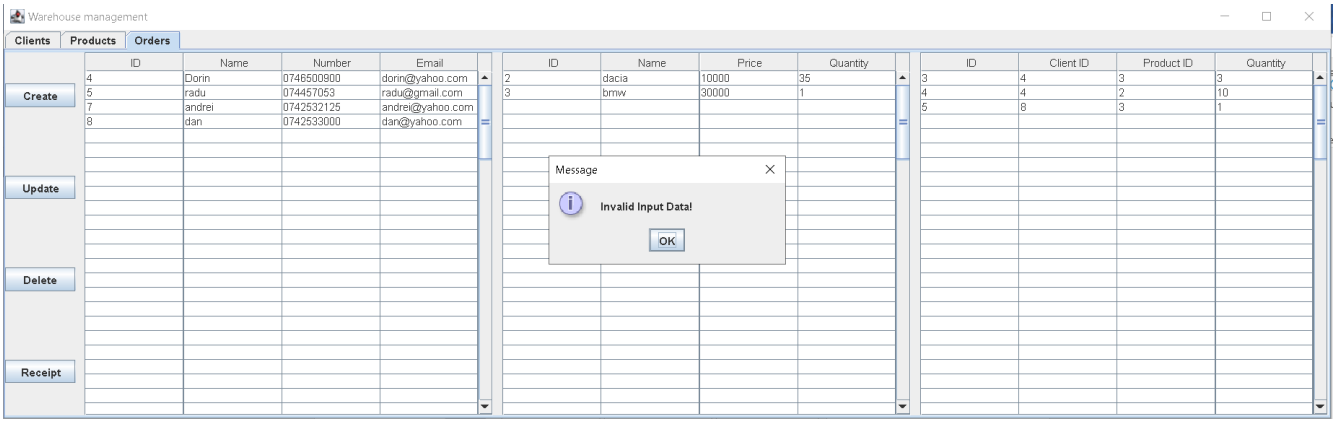


Cazuri de utilizare:

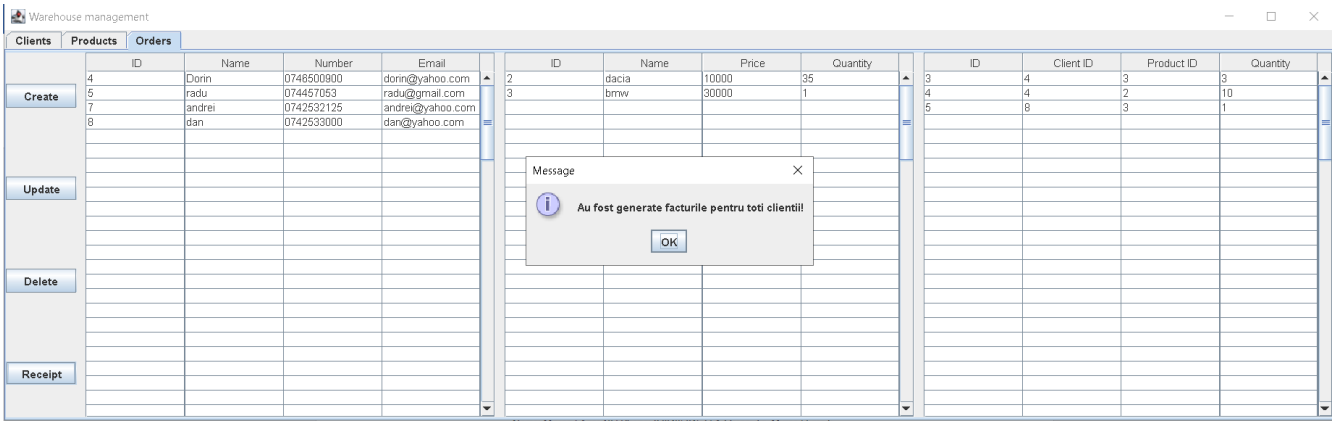
Initial se selecteaza tab-ul dorit (Clients, Products, Orders), iar apoi se alege operatia care urmeaza sa fie executata (Insert, Update, Delete, Receipt). Odata cu apasarea butonului pentru realizarea operatiei, se va deschide o fereastra ce contine casete de text pentru introducerea datelor de catre utilizator.







Aplicatia ofera posibilitatea de a crea facturi pentru comenzile tuturor clientilor. Aceste facturi se regasesc in fisierul facturi.txt .



3. Proiectare

Implementarea OOP a aplicației propriu-zise constă în utilizarea modelului Layered Architecture (Business layer, Data Access Layer, Presentation Layer, model), pentru o proiectare concisă și o structurare a tuturor datelor în mod uniform.

1. Pachetul **Business** cuprinde trei. Clasele sunt **ClientBLL**, **OrderBLL** si **ProductBLL**.
    - Aceste clase realizeaza defapt legatura dintre pachetul **Presentation** si operatiile pe baza de date propriu zisa. Concret, fiecare clasa de tipul BLL au un continut similar(exemplu pentru ClientBLL si OrderBLL):
        - Metode:
            - Public void deleteOrder();
            - Public boolean **updateOrder()**;
            - Public Object[][] **provideData ()**;
            - Public void **generateReceipt**()
            - Public ClientDAO getClientDAO()
2. Pachetul **Connection** conține strict realizarea conexiunii dintre limbajul de dezvoltare Java si baza de date mySql.
3. Pachetul **DataAccess** contine toate operatiile propriu-zise realizate pe baza de date. Acest pachet contine o clasa generica, implementata prin metoda reflectiei, pentru a nu avea sectiuni de cod duplicat care fac o actiune similara pentru toate tabelele in parte.
    - Clasa **AbstractDAO**, utilizata pentru realizarea tututor operatiilor comune pe fiecare tabela din baza de date.
        - Aceasta contine urmatoarele metode:
            - Public **AbstractDAO**();
            - Public void **createSelectQuery**();
            - Public List<> **findAll**();
            - Public String **findById**();
            - Private void **insert**();
            - Private void **update**();
            - Private void **delete**();
        - Urmatoarele campuri private:
            - Private final Class<T> **type –** utilizat pentru a determina clasa fiecarei tabele;
    - Clasele **ClientDAO**, **OrderDAO** si **ProductDAO**, care implementeaza clasa **AbstractDAO,** fiecare avand ca generic modellul cu acelasi nume. Acestea sunt folosite in principiu pentru a suprascrie metode din **AbstractDAO,** in cazul in care se doreste o functionalitate in plus pe langa ceea ce este deja realizat in clasa superclasa.
4. Pachetul **Model,** contine clasele **Client**, **Order**, **Product,** care reprezinta o interpretare a tabelelor din baza de date, cu aceleasi nume de campuri ca in tabelele din baza de date Toate clasele contin accesoare si mutatoare.
5. Pachetul **Presentation,** contine clasele **Controller**, **precum si clasele ce definesc interfata grafica a apicatiei,** utilizata pentru a lega operatiile disponibile in interfata de implementarea opreatiilor pe baza de date.
    - Din clasa **Controller** se remarca urmatoarele metode:
        - public static void **updateObject**(Class classBLL, Class classModel, List<JTextField> textFields) throws Exception;
        - public void addListener();
        - public void updateClientTable()
        - public void updateProductTable()
        - public void updateOrdertTable()
        - public void addListeners()
        - public void UpdateButton()
        - public void DeleteButton()

Diagrama UML a pachetelor:

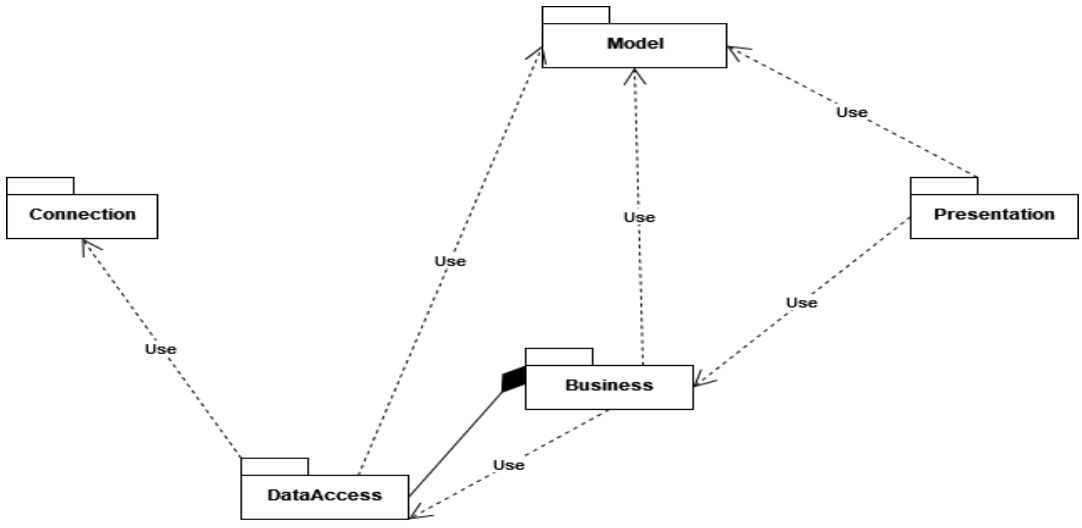
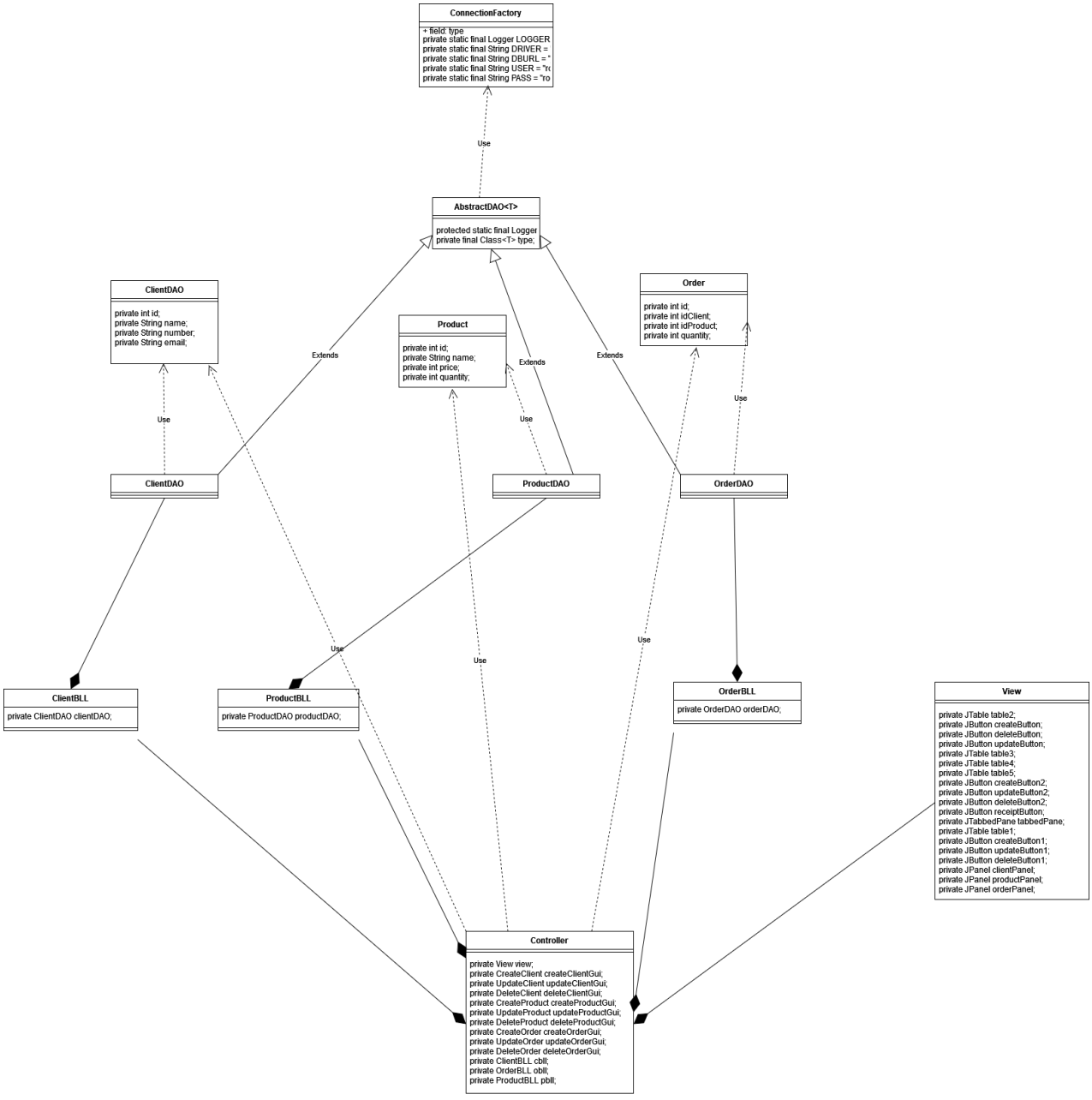


Diagrama UML a claselor:



4. Implementare

Descrierea metodelor importante din unele clase:

- În clasa **controller**:
  - Metoda**addListeners**(...) – aceasta metoda are rolul de a seta cate un action Litsener prentru fiecare buton disponibil in cadrul interfetei grafice aaplicatiei pentru gestiunea bazei de date.
  - Metoda **createOrderButton(**...) – este utilizata pentru a insera o comanda in tabela Order astfel: Utilizatorul selecteaza o coloana din tabela Client si o coloana din tabela Product. Datele din aceste coloane sunt transmise mai departe la obiectul order din aceeasi metoda. Se creeaza obiectul conform cantitatii introduse de utilizator („quantity"), celelalte date ale obiectului fiind deja cunoscute datorita selectiei anterioare. Se apeleaza insertOrder din Business Layer-ul corespunzator si se realizeaza inserarea in tabela.
  - Metoda **printReceiptButton**(...) – este utilizata pentru a genera facturile pentru fiecare client, datele de factura fiind scrise in fisierul facturi.txt
  - Metoda **updateTables**(...) – este utilizata pentru a actualiza datele tabelelor din GUI. Aceasta actualizare se foloseste de fiecare data cand se produc modificari asupra bazei de date

- În clasa **AbstractDAO:**
  - Metoda **update**(...) – este o metoda generica utilizata pentru a transmite o instructiune de update catre baza de date pe coloana creata cu ajutorul obiectului t transmis ca parametru.
  - Metoda **findById**(...) – este o metoda generica utilizata pentru a gasi o tuplda cu un id specificat in continutul tabelelor bazei de date;

```
• public String generateReceipt(int id, ProductDAO productDAO, OrderDAO
  orderDAO)
  {
      String ret = "";
      Product product = null;
      Client client = clientDAO.findById(id);
      List<Order> orders = new ArrayList<Order>();

      for (Order o : orderDAO.findAll())
          if (o.getIdClient() == client.getId())
              orders.add(o);

      if (orders.isEmpty()) return ret;

      ret += "Factura pentru " + client.getName() + " : \n";

      int suma = 0;

      for (Order o : orders)
      {
          product = productDAO.findById(o.getIdProduct());
          ret += product.getName() + " - " + o.getQuantity() + " buc - "
  + product.getPrice() * o.getQuantity() + ";\n";
          suma += product.getPrice() * o.getQuantity();

      }

      ret += "Total: " + suma + "\n";
      return ret;
  }
```

```
• public Object[][] provideData()
  {
      int i = 0;
      Object[][] ret =  new Object[100][4];
      List<Client> clients = clientDAO.findAll();
      for (Client c : clients)
      {
          ret[i][0] = c.getId(); ret[i][1] = c.getName(); ret[i][2] =
  c.getNumber(); ret[i][3] = c.getEmail();
          i++;
      }


      return ret;
  }
• public boolean insertOrder(Order order, ProductDAO productDAO)
  {
      Product p = productDAO.findById(order.getIdProduct());
      if (order.getQuantity() > p.getQuantity())
          return false;

      p.setQuantity(p.getQuantity() - order.getQuantity());

      orderDAO.insert(order);
      productDAO.update(order.getIdProduct(), p);
      return true;
  }
• public boolean updateOrder(int idOrder, Order order, ProductDAO
  productDAO)
  {
      Order old =  orderDAO.findById(idOrder);
      Product p = null;

      if (old.getIdProduct() == order.getIdProduct())
      {
          p = productDAO.findById(order.getIdProduct());
          if (order.getQuantity() - old.getQuantity() > p.getQuantity())
              return false;

          p.setQuantity(p.getQuantity() + (old.getQuantity() -
  order.getQuantity()));
          productDAO.update(p.getId(), p);
          orderDAO.update(idOrder, order);

      }
      else
      {
          Product old_p = productDAO.findById(old.getIdProduct());
          old_p.setQuantity(old_p.getQuantity() + old.getQuantity());
          p = productDAO.findById(order.getIdProduct());

          if (p.getQuantity() < order.getQuantity())
              return false;

          p.setQuantity(p.getQuantity() - order.getQuantity());
```

```
            productDAO.update(old_p.getId(), old_p);
            productDAO.update(p.getId(), p);
            orderDAO.update(idOrder, order);
        }
        return true;
    }
    public void deleteOrder(int idOrder, ProductDAO productDAO)
    {
        Order o = orderDAO.findById(idOrder);
        Product p = productDAO.findById(o.getIdProduct());
        p.setQuantity(p.getQuantity() + o.getQuantity());
        productDAO.update(p.getId(), p);
        orderDAO.delete(idOrder);
    }
    public void orderCreateButton()
    {
        try
        {
            Order order = new Order();

    order.setIdClient(Integer.parseInt(createOrderGui.getClientIdTextField(
    ).getText()));

    order.setIdProduct(Integer.parseInt(createOrderGui.getProductIdTextFiel
    d().getText()));

    order.setQuantity(Integer.parseInt(createOrderGui.getQuantityTextField(
    ).getText()));

            if (!obll.insertOrder(order, pbll.getProductDAO()))
            {
                JOptionPane.showMessageDialog(view, "Invalid Input Data!");
                return;
            }
            updateTables();
            createOrderGui.setVisible(false);
        }
        catch (Exception e) { JOptionPane.showMessageDialog(view, "Invalid
    Input Data!");}
    }
    public void orderUpdateButton()
    {
        try
        {
            int aux_id;
            Order order = new Order();
            aux_id =
    Integer.parseInt(updateOrderGui.getIdTextField().getText());

    order.setIdClient(Integer.parseInt(updateOrderGui.getClientIdTextField(
    ).getText()));

    order.setIdProduct(Integer.parseInt(updateOrderGui.getProductIdTextFiel
    d().getText()));

    order.setQuantity(Integer.parseInt(updateOrderGui.getQuantityTextField(
```

```
    ).getText()));

            if (obll.getOrderDAO().findById(aux_id) == null)
            {
                JOptionPane.showMessageDialog(view, "Invalid Input Data!");
                return;
            }

            if (!obll.updateOrder(aux_id, order, pbll.getProductDAO()))
            {
                JOptionPane.showMessageDialog(view, "Invalid Input Data!");
                return;
            }

            updateTables();
            updateProductGui.setVisible(false);
        }
        catch (Exception e) { JOptionPane.showMessageDialog(view, "Invalid
    Input Data!");}

    }
•   public void orderDeleteButton()
    {
        try
        {
            int aux_id =
    Integer.parseInt(deleteOrderGui.getIdTextField().getText());

            if (obll.getOrderDAO().findById(aux_id) == null)
            {
                JOptionPane.showMessageDialog(view, "Invalid Input Data!");
                return;
            }

            obll.deleteOrder(aux_id, pbll.getProductDAO());
            updateTables();
            deleteOrderGui.setVisible(false);
        }
        catch (Exception e) {JOptionPane.showMessageDialog(view, "Invalid
    Input Data!");}
    }
•   public void printReceiptButton() {
        List<Client> clients = cbll.getClientDAO().findAll();

        File file = new File("facturi.txt");
        PrintWriter writer = null;

        try {
            writer = new PrintWriter(file);
        } catch (Exception ignored) {}

        for (Client c : clients)
        {
            System.out.println(cbll.generateReceipt(c.getId(),
    pbll.getProductDAO(), obll.getOrderDAO()));
            writer.print(cbll.generateReceipt(c.getId(),
```

```
    pbll.getProductDAO(), obll.getOrderDAO()));
        }

        writer.close();
        JOptionPane.showMessageDialog(view, "Au fost generate facturile
    pentru toti clientii!");
    }
    public void addListeners()
    {
        view.getCreateButton1().addActionListener(e ->
    clientShowCreateButton());
        createClientGui.getAcceptButton().addActionListener(e ->
    clientInsertButton());
        view.getUpdateButton1().addActionListener(e ->
    clientShowUpdateButton());
        updateClientGui.getAcceptButton().addActionListener(e ->
    clientUpdateButton());
        view.getDeleteButton1().addActionListener(e ->
    clientShowDeleteButton());
        deleteClientGui.getAcceptButton().addActionListener(e ->
    clientDeleteButton());

        view.getCreateButton().addActionListener(e ->
    productShowCreateButton());
        createProductGui.getAcceptButton().addActionListener(e ->
    productCreateButton());
        view.getUpdateButton().addActionListener(e ->
    productShowUpdateButton());
        updateProductGui.getAcceptButton().addActionListener(e ->
    productUpdateButton());
        view.getDeleteButton().addActionListener(e ->
    productShowDeleteButton());
        deleteProductGui.getAcceptButton().addActionListener(e ->
    productDeleteButton());

        view.getCreateButton2().addActionListener(e ->
    orderShowCreateButton());
        createOrderGui.getAcceptButton().addActionListener(e ->
    orderCreateButton());
        view.getUpdateButton2().addActionListener(e->
    orderShowUpdateButton());
        updateOrderGui.getAcceptButton().addActionListener(e ->
    orderUpdateButton());
        view.getDeleteButton2().addActionListener(e ->
    orderShowDeleteButton());
        deleteOrderGui.getAcceptButton().addActionListener(e ->
    orderDeleteButton());

        view.getReceiptButton().addActionListener(e ->
    printReceiptButton());
    }
```

5. Concluzii

In concluzie, aceasta tema mi-a dezvoltat capacitatea de a crea aplicatii ce gestioneaza baze de date si m-a ajutat sa ma familiarizez cu conceptul de tehnica de reflexie.

6.Bibliografie

1. https://dsrl.eu/courses/pt/materials/A3_Support_Presentation.pdf
2. https://www.thoughtco.com/using-java-naming-conventions-2034199
3. https://docs.oracle.com/javase/tutorial/uiswing/
4. https://itextpdf.com/en
5. https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
6. https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html
7. https://jenkov.com/tutorials/java-reflection/index.html
8. https://dzone.com/articles/layers-standard-enterprise