



Rapport: Mini Projet P00

Gestion des commandes

Encadré par:

- Mr. Noredine Gherabi

Réalisé par:

- Hrouch Aymane
- Ghoundal Youssef

Sommaire:

- Remerciements
- Introduction
- Outils Utilisés
- Architecture
- Présentation des classes
- Conclusion

Remerciements:

On tient tout d'abord à remercier notre encadrant Mr. Gherabi Noredine et
Nous souhaitons à exprimer notre gratitude aussi à avoir donné l'occasion pour
faire un projet qui nous a permet d'acquérir des nouvelles connaissances et de
développer nos compétences.

Introduction:

La gestion des données est un critère essentiel pour toute entreprise ou établissement il se peut que ça soit une gestion de stock, gestion des ressources humaines ...

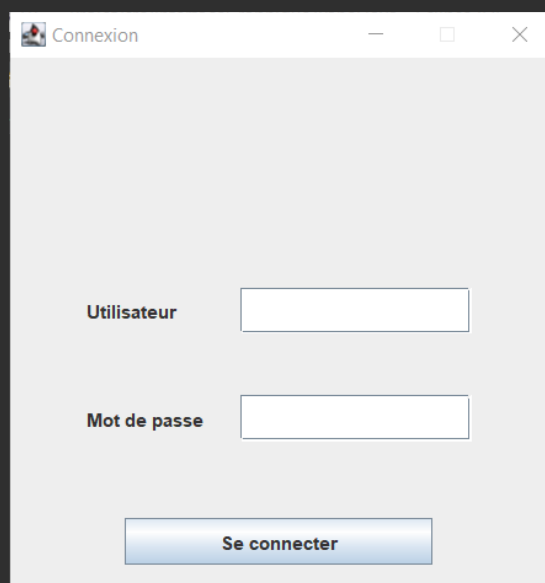
De plus, la construction d'une application de gestion est essentielle pour les programmeurs débutants, surtout que ce genre des applications se base sur les opérations fondamentales, connues par le terme **CRUD** (Create, Read, Update, Delete). Pour cela nous avons utilisé nos connaissances acquis lors les deux modules **programmations Orientés objet Java** et **Système d'info et bases de données** pour développer une interface utilisateur qui va aider une société de construction industrielle pour gérer les commandes de leurs clients.

Cette application permet d'ajouter, modifier et supprimer les clients, produits, commandes, livraisons et factures, sécurisé par **authentification** évidemment.

Nous avons décidé que la meilleure approche est d'utiliser des **onglets**.

Avant de commencer la présentation des différents composants de notre réalisation, voilà deux capture d'écran de la résultats finale:

Fenêtre d'authentification



Connexion

Utilisateur

Mot de passe

Se connecter

Interface Principale

numeroclient	Nom	Prenom	Adresse	Telephone
66	Hrouch	Aymane	Khenifra	0637478085
67	Ghoundal	Youssef	Beni-Mellal	0654357649

Pour modifier ou supprimer, l'utilisateur doit remplir tous les champs (champ de la clé primaire au-dessous inclus).

Pour ajouter, il suffit de remplir les champs au-dessus.

Outils Utilisés:

Pour réaliser ce projet nous avons utilisé les outils suivants:

- WampServer (MySQL)
- Eclipse (IDE)
- Le langage SQL

- Java (Swing)

Architecture:

Dans ce chapitre, nous présentons l'architecture sur laquelle nous avons développé notre application.

Nous avons utilisé deux packages:

1. entitees

Contient les classes java définies dans le diagramme de classes.

- Client.java
- Produits.java
- Commande.java
- Livraison.java
- Facture.java

2. InterfaceGraphique

Contient,

une classe pour gérer la connectivité entre notre application et la base de données:

- DB.java

une classe qui fournit des méthodes qu'on utilise souvent dans les autres classes de ce package.

- Util.java

les classes d'interfaces graphiques:

- LoginFrame.java
- MainFrame.java
- Panel.java
- InterfaceClient.java
- InterfaceProduits.java
- InterfaceCommande.java
- InterfaceLivraison.java
- InterfaceFacture.java

Présentation des classes:

DB.java:

Cette classe contient une méthode `demarrer()` qui établit une connexion avec notre base de données:

```
public static void commencer() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con =
        DriverManager.getConnection("jdbc:mysql://localhost:3306/poo_db","root","")
        ;
        stmt = con.createStatement();
    } catch(ClassNotFoundException ex) {
        Util.afficherErreur("Erreur lors de chargement de drive: " +
ex.getMessage());
        System.exit(1);
    }
    catch(SQLException ex) {
        Util.afficherErreur(ex.getMessage());
        System.exit(1);
    }
}
```

et deux autres méthodes qui permettent d'exécuter les commandes SQL en capturant les exceptions , `executeUpdate(String req)` et `executeQuery(String req)`:

```
public static int executeUpdate(String req) {
    try {
        return stmt.executeUpdate(req);
    } catch (SQLException e) {
        Util.afficherErreur("Erreur lors de l'execution de la
requete" + e.getMessage());
        return -1;
    }
}

public static ResultSet executeQuery(String req) {
    try {
        return stmt.executeQuery(req);
    } catch (SQLException e) {
        Util.afficherErreur("Erreur lors de l'execution de la
commande SQL" + e.getMessage());
        return null;
    }
}
```

Util.java:

Cette classe contient des méthodes qui affiche des fenêtres d'erreur ou d'information, ça nous permet d'écrire un code qui est plus court, significatif et clair.

```
public class Util {
    public static void afficherErreur(String message)
    {
        JOptionPane.showMessageDialog(null, message, "Erreur!",
JOptionPane.ERROR_MESSAGE);
    }

    public static void afficherInfo(String message) {
        JOptionPane.showMessageDialog(null, message, "Alert!",
JOptionPane.INFORMATION_MESSAGE);
    }

    public static void afficherInfo(String message, String titre) {
```



```

        JOptionPane.showMessageDialog(null, message, titre,
JOptionPane.INFORMATION_MESSAGE);
    }
}

```

LoginFrame.java:

C'est le conteneur qui nous permet de se connecter à notre application en utilisant un login et un mot de passe, c'est une classe qui hérite de `JFrame` et implimente `ActionListener`

```
public class LoginFrame extends JFrame implements ActionListener
```

Elle utilise les attributs suivantes:

```

JLabel utilisateurLabel = new JLabel("Utilisateur");
JLabel mdpLabel = new JLabel("Mot de passe");
JTextField utilisateurTextField = new JTextField();
JPasswordField mdpTextField = new JPasswordField();
JButton btn = new JButton("Se connecter");

```

Pour bien organiser,

nous avons créé une méthode `setLocationAndSize()` qui définit l'emplacement et la taille des composants:

```

public void setLocationAndSize() {
    utilisateurLabel.setBounds(50,150,100,30);
    mdpLabel.setBounds(50,220,100,30);
    utilisateurTextField.setBounds(150,150,150,30);
    mdpTextField.setBounds(150,220,150,30);
    btn.setBounds(75,300,200,30);
}

```

et une autre, `ajouterComposants()` qui ajoute les composant au conteneur principale:

```

public void ajouterComposants() {
    this.add(utilisateurLabel);
    this.add(mdpLabel);
    this.add(utilisateurTextField);
    this.add(mdpTextField);
    this.add(btn);
}

```

Et bien sûr la méthode `actionPerformed(ActionEvent e)`

qui s'exécute quand on appuis sur la bouton, elle vérifie l'identité de l'utilisateur, puis si il est légitime, il lance notre principale interface (`new MainFrame().afficher();`).

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == btn) {
        String loginText;
        String pwdText;
        loginText = utilisateurTextField.getText();
        pwdText = new String(mdpTextField.getPassword());
        try {
            ResultSet rs = DB.executeQuery("select login, pwd from users");
            while(rs.next()){
                if(loginText.equals(rs.getString(1)) &&
pwdText.equals(rs.getString(2))) {
                    Util.afficherInfo("Connecté avec succès!");
                    this.setVisible(false);
                    new MainFrame().afficher();
                    return;
                }
            }
            Util.afficherInfo("Login ou mot de passe Invalide.");
        }
        catch(SQLException ex) {
            Util.afficherErreur("Erreur lors de connexion a la BDD de drive: " +
ex.getMessage());
            System.exit(1);
        }
    }
}
```

MainFrame.java:

Voilà le conteneur principale de notre interface qui hérite de `JFrame`, il contient deux composants filles, une barre et un conteneur des onglets.

La barre contient une bouton de déconnexion, les onglets sont les interfaces graphique de gestion.

```
JToolBar barre = new JToolBar();
JButton deconnexion = new JButton("Deconnexion");
JTabbedPane tp = new JtabbedPane();
```

il aussi utilise le même principe comme LoginFrame.java concernant les deux méthodes `setLocationAndSize()` et `AjouterComposants()`, et il y a une autre méthode `afficher()`, qui ajoute les interfaces graphiques de gestion et affiche ce conteneur:

```
public void afficher() {
    tp.add("Clients", new InterfaceClient());
    tp.add("Produits", new InterfaceProduits());
    tp.add("Commandes", new InterfaceCommande());
    tp.add("Livraisons", new InterfaceLivraison());
    tp.add("Factures", new InterfaceFacture());
    this.setVisible(true);
}
```

Pour déconnecter on doit écouter la bouton `deconnexion`, pour cela on utilise la méthode suivante qu'on appelle dans le constructeur:

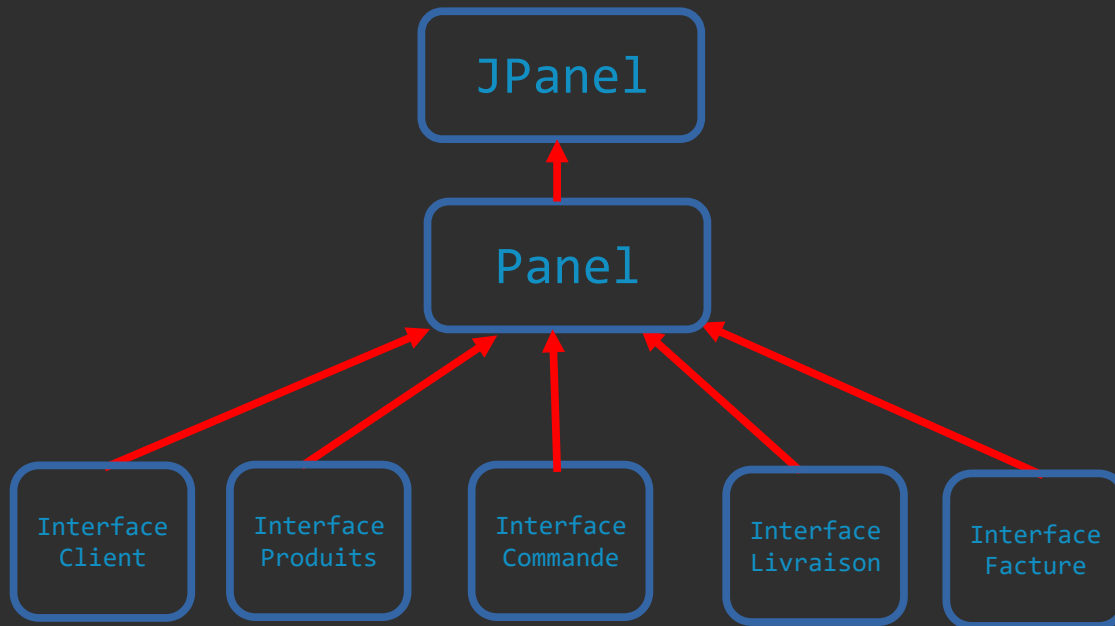
```
public void écouterDeconnexion() {
    deconnexion.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            cacher();
            new LoginFrame();
        }
    });
}
```

La méthode `cacher()`; serve à cacher le conteneur,

```
public void cacher() {
    this.setVisible(false);
}
```

Panel.java:

Une classe abstraite, et probablement la plus importante pour notre interface graphique, par ce que tous les classes des interfaces graphiques de gestion héritent de cette classe.



Extends: →

Dans ce classe il existe les attributs et les méthodes qui seront utilisés par les autres classes, et aussi deux méthodes abstraites qui doit obligatoirement être implémentées dans les autres classes.

Le code contient des commentaires qui donnent une explication précise.

```
public abstract class Panel extends JPanel {
    abstract public void setLabels(); // définir les labels des champs
    abstract public boolean verifier(); // vérifier les champs

    public String idText; // nom de la clé primaire
    String[] tableHeader = null; // tableau contenant les noms des colonnes de
notre tableau
    String tableName = ""; // nom de table au base de données

    JLabel dateFormat = new JLabel("(YYYY-MM-DD)"); // format des dates

    JPanel jPanel1 = new JPanel(), // Panel qui contient les champs
        jPanel2 = new JPanel(); // Panel qui contient le tableau

    /* Tous les classes contient au plus 4 champs,
```

```

*/
    si une classe contient moins que 4 on va utiliser setVisible pour la cacher

    JLabel jLabels[] = new JLabel[4];
    JTextField txtFields[] = new JTextField[4];

    JLabel idLabel = new JLabel();
    JTextField idTF = new JTextField();

    // Tous les classes contient ces boutons
    JButton
    btnAjouter = new JButton("Ajouter"),
    btnModifieur = new JButton("Modifieur"),
    btnSupprimer = new JButton("Supprimer");
    JButton clearBtn = new JButton("X");

    // Tableau
    JTable tb = new JTable();
    JScrollPane sp = new JScrollPane();

    DefaultTableModel model = null;

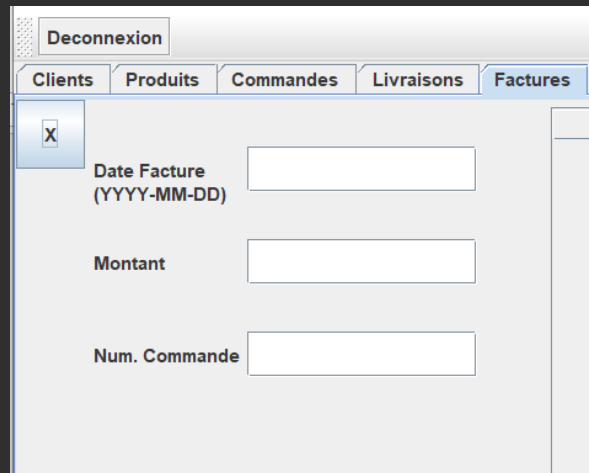
    public Panel() {

        // initialiser les elements des deux tableaux jLabels[] et txtFields[]
        for(int i=0; i < jLabels.length; i++) {
            jLabels[i] = new JLabel();
            txtFields[i] = new JTextField();
        }

        setLabels();
        ajouterComposants();
        setLocationAndSize();
        EcouterBoutons();
    }

```

-
- `dateFormat` n'est visible que pour les interfaces qui possèdent un champ d'une date, `Facture` par exemple. Donc dans cette classe on va utiliser `dateFormat.setVisible(true)` et la format va être affiché au-dessous du premier champ:



- **EcouterBoutons()** est une méthode qui permet d'écouter les quatre boutons **Ajouter**, **Modifier**, **Supprimer** et «X». Les classes qui héritent cette classes doit la **redéfinir**. Mais on a l'initialisé ici quand même, par ce que l'écoute de les bouton **Supprimer** et «X» peuvent être généraliser.

Pour Supprimer, la logique est de vérifier le champ de la clé primaire puis exécuter une commande DELETE de SQL puis supprimer la ligne depuis le tableau sur l'interface graphique.

Pour «X» on appelle la méthode **clearTextFields** qui nettoie les champs.

```
public void EcouterBoutons() {
    btnSupprimer.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            String id = idTF.getText();
            if(!verifierId(id , idText)) return;
            String requete = "DELETE FROM "
                            + tableName + " WHERE "
                            + idText + "=" + id;

            int y = getRow(id);
            if(y == -1) {
                Util.afficherInfo(idText + " Inexistant");
                return;
            }
            model = (DefaultTableModel) tb.getModel();
            model.removeRow(y);
            DB.executeUpdate(requete);
            idTF.setText("");
        }
    });
}
```

```

clearBtn.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        clearTextFields();
    }

});
}

```

On a aussi déclarer d'autres méthodes qui sont utilisables dans toutes les autres classes qui héritent de cette classe, par exemple des méthodes qui concernent le tableau:

```

public void initTableau(String th[])
public void écouterTableau(JTable table)
public void chargerTableau(int tableauLength, String requete)

```

Des méthodes qui gèrent les lignes de tableau:

```

public void ajouterLigne(int tableauLength, String requete)
public void modifierLigne(String[] objet, int ligne)

```

Une méthode qui vérifie le champ de la clé primaire:

```

public boolean verifierId(String id, String label)

```

Autre méthodes utiles:

```

public void donneesIncompleteFenetre()
public int getRow(String id)
public void clearTextFields()

```

InterfaceClient.java:

InterfaceProduits.java:

InterfaceCommande.java:

InterfaceLivraison.java:

InterfaceFacture.java:

Ces classes ont la même structure et logique, c'est pas la peine de les présenter séparément. On va parler juste de **InterfaceClient.java** comme un exemple.

D'abord on déclare un objet de la classe Client

```

public class InterfaceClient extends Panel {
    Client c1 = new Client();
}

```

Dans le constructeur on appelle `super()`, puis on définit les attributs que nous avons déclaré dans `Panel1`, puis on initialise et charge le tableau

```
public InterfaceClient() {
    super();
    tableName = "client";
    idText = "numeroclient";
    tableHeader = new String[] { idText, "Nom", "Prenom", "Adresse",
"Telephone"};
    initTableau(tableHeader);
    idLabel.setText(idText + ": ");
    chargerTableau(tableHeader.length, "SELECT * FROM client ORDER BY " +
idText);
}
```

On définit les méthode `setLabels()`, `verifier()` et `remplirClient()`

```
public void setLabels() {
    jLabels[0].setText("Nom");
    jLabels[1].setText("Prénom");
    jLabels[2].setText("Adresse");
    jLabels[3].setText("Téléphone");
}

public boolean verifier() {
    if( txtFields[0].getText().isEmpty() ||
        txtFields[1].getText().isEmpty() ||
        txtFields[2].getText().isEmpty() ||
        txtFields[3].getText().isEmpty()
    ) {donneesIncompleteFenetre();
return false;
}

    if(!txtFields[3].getText().matches("\\d{10}")) { // Si le telephone
contenir quelque chose autre que des nombres.
        Util.afficherInfo("Telephone doit contenir 10 nombres.", "Nombre de
telephone invalide.");
        return false;
    }
    return true;
}

public void remplirClient() {
    cl.nom = txtFields[0].getText();
    cl.prenom = txtFields[1].getText();
    cl.adresse = txtFields[2].getText();
    cl.telephone = txtFields[3].getText();
}
```


`verifier()` serve a vérifier les champs, ici pour client on doit vérifier que tous les champs sont non vides et que le nombre de téléphone contient 10 nombres.

Un autre exemple pour les **produits**: on doit vérifier que les champs sont non vides et que le champs de quantité est un nombre entier, et le champs de prix est un nombre réel.

`RemplirClient()` serve a remplir l'objet qu'on a déclaré plus tôt, pour les autres classes il existe une méthode similaire qui fait le même pour l'objet correspondant, par exemple la méthode `RemplirCommande()` dans la classe `InterfaceCommande` et `RemplirFacture()` dans la classe `InterfaceFacture`.

Finalement, on redéfinit la méthode `EcouterBoutons()`,

Mais on doit également appeler la méthode de la superclasse. Donc on fait appelle à `super.EcouterBoutons` puis on écoute les deux boutons Ajouter et Modifier.

Essentiellement ce que nous faisons est de vérifier que les champs sont valides, si oui on remplit notre objet on utilisant la fonction de remplissage (`RemplirClient()` ici) puis on exécute la commande SQL pour ajouter/modifier les données dans la base de données en utilisant les méthodes de la classe `DB`, finalement on appelle une des méthodes qui va montrer notre changement à l'interface graphique

`ajouterLigne(int tableauLength, String requete)` ajoute les données à notre table + vider les champs.

`modifierLigne(String[] objet, int ligne)` modifie le ligne correspondant.

Voilà le code pour la classe `InterfaceClient`

```
public void EcouterBoutons() {
    super.EcouterBoutons();
    btnAjouter.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if(!verifier()) return;
        }
    });
}
```

```

        remplirClient();
        // Remplir les information de client.
        String requete = "INSERT INTO
Client(nom,prenom,adresse,telephone)"
        + "VALUES('" + cl.nom + "', '" + cl.prenom + "', '"
+
        cl.adresse + "', '" + cl.telephone + "')";
        if(DB.executeUpdate(requete) == -1) return;

        ajouterLigne(tableHeader.length, "SELECT * FROM client ORDER BY
numeroclient DESC LIMIT 1");
    }
});

btnModifieur.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        model = (DefaultTableModel) tb.getModel();
        String id = idTF.getText();
        if(!verifierId(id, idText) || !verifier()) return;
        int y = getRow(id);
        if(y == -1) {
            Util.afficherInfo(idText + " Inexistant");
            return;
        }
        remplirClient();
        String requete = "UPDATE client"
            + " SET nom= '" + cl.nom + "'"
            + ", prenom= '" + cl.prenom + "'"
            + ", adresse= '" + cl.adresse + "'"
            + ", telephone= '" + cl.telephone + "'"
            + " WHERE numeroclient=" + id;

        if(DB.executeUpdate(requete) == -1) return;

        modifierLigne(new String[] {id, cl.nom, cl.prenom, cl.adresse,
cl.telephone}, y);
    }
});
}

```

Conclusion:

A travers ce projet. Nous avons découvert de nouvelles informations et réussi à produire notre programme. Nous avons solidifié les notions de la programmation orienté objet et bien compris ces concepts, ce qui absolument va nous aider au future.

Ce rapport est accompagné avec

- Code java
- un fichier .sql pour importer en utilisant phpmyadmin, le nom de la base de données est 'projet_poo'.