

# Rapport du projet POO

“ Cabinet médicale ”



# travail fait par :

**Nom :**

**Prénom :**

**Matricule :**

<b>Mearkchi</b>	<b>Rahma</b>	<b>222231657517</b>
<b>Boumahra</b>	<b>nada</b>	<b>222231452406</b>
<b>Bencheikh</b>	<b>Aya</b>	<b>222231363701</b>
<b>Sayadi</b>	<b>Chaima</b>	<b>212231446507</b>

# Sommaire

-01-

**Introduction**

-02-

**les ressources utilisées**

-03-

**Notion et logiciel utilisé**

-04-

**le fonctionnement globale  
de l'application**

-05-

**Analyse du code**

-06-

**La base de données**

-07-

**Les interfaces  
graphiques**

-08-

**Code java simple**

-09-

**Conclusion**

# 01

## Introduction

Dans un cabinet médical , il n'est pas toujours facile de gérer les différentes prises de rendez vous , répondre au téléphone , rappeler les heures des rendez-vous , fixer des rendez-vous.....,etc.

Ce projet JAVA a pour objectif de proposer une solution logicielle complète pour la gestion quotidienne d'un cabinet médical. Notre motivation principale réside dans le désir de faciliter le travail des médecins, secrétaires et autres personnels du cabinet, tout en améliorant la prise en charge des patients.

# 02

## Les ressources utilisées

- Enregistrement d'une vidéo explicative (de notre prof de TP de BDD) :

<https://drive.google.com/file/d/1WgEPrGGzFl8i4QV9kiGsTZmJKlxLzPcE/view>

- Support de TP :

[https://drive.google.com/file/d/1qK\\_LBCR5TwpRB8HLVpcoexPPcjr1oxqz/view](https://drive.google.com/file/d/1qK_LBCR5TwpRB8HLVpcoexPPcjr1oxqz/view)

- Chaine Youtube 1 :

Khadem tech

- Chaine youtube 2 :

Knowledge to Share

- Photo utiliser dans le projet :

[https://www.freepik.com/free-psd/3d-cartoon-men-illustration\\_24770589.htm?query=hospital%20mascot%203d#from\\_view=detail\\_alsolike](https://www.freepik.com/free-psd/3d-cartoon-men-illustration_24770589.htm?query=hospital%20mascot%203d#from_view=detail_alsolike)

03

# Notion et logiciel utilisé

Pour notre projet de développement d'une petite application, on a utilisé plusieurs outils et technologies. On a choisi **Eclipse** et **intelliJ idea** avec le plugin **WindowBuilder**( pour créer facilement des **interfaces graphiques** en utilisant des fonctionnalités comme le glisser-déposer et la modification visuelle des composants. Pour gérer les données, on a utilisé **JDBC** Oracle pour connecter notre application à une base de données **Oracle** et **SQL\*Plus** pour exécuter des **requêtes SQL** et gérer la **base de données**.

On a également appliqué les principes de la programmation **orientée objet (OOP)** en Java, comme , **les classes** ,**les interfaces** et **les exceptions** pour structurer notre code de manière modulaire et réutilisable. Ces outils et concepts nous a permis de développer une application **fonctionnelle** et **bien organisée**.

# 04

## Le fonctionnement Globale l' application

### 1. Fenetre de connexion :

Pour accéder à l'application de **gestion du cabinet médical**, l'utilisateur doit d'abord se connecter en utilisant un **nom d'utilisateur** et un **mot de passe**. Une fois connecté, il aura accès à diverses fonctionnalités de gestion, telles que la gestion des patients, des dossiers de patients et des rendez-vous.

### 2. Fenetre de gestion de rendez vous :

Dans la fenêtre "**Rendez-vous**", l'utilisateur (**secrétaire**) peut planifier des visites chez le médecin en précisant le **nom**, **le prénom**, ainsi que **la date** et **l'heure** du rendez-vous en fonction de la disponibilité du médecin. Une fois la date de rendez-vous sélectionnée, l'utilisateur peut sauvegarder le rendez-vous et même le supprimer.

# 04

## Le fonctionnement Globale l' application

### **3. Fenêtre de gestion des patients :**

Dans la fenêtre "**Patients**", l'utilisateur ( **médecin** ) peut ajouter, modifier ou supprimer des patients. Ils a la main de saisir les informations suivantes : **Nom, Prénom, Téléphone, Antécédents, Ordonnance et Observations**. Ensuite, il doit cliquer sur le bouton "Sauvegarder" pour enregistrer les données.

Une fois l'insertion effectuée, l'utilisateur a la possibilité de modifier les informations d'un patient ou de le supprimer . Cette fonctionnalité facilite la gestion des patients du cabinet médical.

---

### **4. Fenêtre de gestion des fiches patients :**

Après avoir sauvégarde les informations des patients dans la fenêtre "**Patients**", ces informations seront automatiquement enregistrées dans une "**Fiche-patients**". Pour accéder à la fiche d'un patient, il suffit de saisir le nom et le prénom du patient, puis de cliquer sur le bouton "**Chercher**".

L'utilisateur (médecin) a également la possibilité d'imprimer cette fiche pour la remettre au patient dans le cas où il l'a demandé ou de la supprimer si nécessaire.

# 05

# Analyse du code

## Packages importés:

- **javax.swing.\***: Composants GUI Swing pour créer l'interface utilisateur.
- **javax.swing.table.DefaultTableModel**: Modèle de table par défaut pour les tableaux Swing.
- **java.awt.\***: Classes AWT pour la création de l'interface graphique.
- **java.awt.event.\***: Gestion des événements AWT.
- **java.awt.print.\***: Classes pour la gestion de l'impression.
  - a. **java.text.SimpleDateFormat**: Pour formater les dates.
- **java.util.\***: Contient les classes utilitaires comme **ArrayList**, **Calendar** et **HashMap**.
- **java.awt.image.BufferedImage**: Classe pour les images mémoire.
- **java.io.\***: Gestion des entrées/sorties, notamment pour les fichiers.
- **javax.imageio.ImageIO**: Pour lire et écrire des images.
- **java.sql.\***: Classes pour la gestion des bases de données SQL.
- Ces importations couvrent une large gamme de fonctionnalités, indiquant que le programme a des interfaces utilisateur complexes, des interactions avec une base de données, et peut gérer l'impression et les images.

```
*LoginGUI.java x
1 package projet2;
2 import javax.swing.*;
3 import javax.swing.table.DefaultTableModel;
4
5 import java.awt.*;
6 import java.awt.event.ActionEvent;
7 import java.awt.event.ActionListener;
8 import java.awt.print.PageFormat;
9 import java.awt.print.Printable;
10
11 import java.text.SimpleDateFormat;
12 import java.util.ArrayList;
13 import java.util.Calendar;
14 import java.util.HashMap;
15
16 import java.awt.image.BufferedImage;
17 import java.io.File;
18 import java.io.IOException;
19
20 import javax.imageio.ImageIO;
21 import java.sql.*;
22
23 import java.awt.print.PrinterException;
24 import java.awt.print.PrinterJob;
25
26
```

## La classe LoginGUI :

- **LoginGUI:** Classe principale qui implémente une interface graphique de connexion.
- **implements ActionListener:** Permet à la classe de réagir aux événements d'action, comme les clics sur les boutons

```
public class LoginGUI implements ActionListener {
```

### 1. Attributs:

- Les **JLabel**, **JTextField**, **JPasswordField**, **JButton**, et **JFrame** sont utilisés pour créer l'interface utilisateur.
- **HashMap<String, String> users:** Contient les informations d'identification des utilisateurs.
- **ArrayList<String> patientList:** Stocke les informations des patients.
- **JTable patientTable** et **JTable table:** Utilisés pour afficher des informations sous forme de tableaux.
- **Connection** et **Statement:** Utilisés pour les interactions avec une base de données SQL.

```
30  private JLabel cabinetLabel;
31  private JLabel usernameLabel;
32  private JTextField usernameTextField;
33  private JLabel passwordLabel;
34  private JPasswordField passwordTextField;
35  private JButton loginButton;
36  private JButton clearButton;
37  private JLabel successLabel;
38  private JLabel backgroundLabel;
39  private HashMap<String, String> users;
40  private JFrame secondFrame;
41  private JTable patientTable; // Table to display patient information
42  private JTextField nameField;
43  private JTextField lastNameField;
44  private JTextField telField;
45  private ArrayList<String> patientList;
46  private JTable table; // Table for Rendez Vous
47  private Object antecedents;
48
49  private static Connection connection;
50  private Statement statement;
51
```

## La classe LoginGUI :

### 2. Le Constructeur LoginGUI :

- Initialisation des utilisateurs :

**users.put():** Ajoute des paires **nom d'utilisateur/mot de passe** dans le **HashMap**.

```
public LoginGUI() {  
    users = new HashMap<>();  
    users.put("boumahra", "nada2003");  
    users.put("sayadi", "chaima04");  
    users.put("merakchi", "1234");  
    users.put("benchikh", "aya.01");  
    users.put("a", "a");  
}
```

- Crée une **ArrayList** pour **patientList** :

```
patientList = new ArrayList<>();
```

- Configuration du panneau de connexion :

```
JPanel panel = new JPanel();  
JFrame frame = new JFrame("se connecter");  
frame.setSize(736, 1104);  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
frame.add(panel);  
  
panel.setLayout(null);
```

-**JPanel panel:** Panneau principal pour organiser les composants.

-**JFrame frame:** Fenêtre principale pour l'interface de connexion.

-**frame.setSize(736, 1104):** Définit la taille de la fenêtre.

-**frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE):** Ferme l'application lorsque la fenêtre est fermée.

-**panel.setLayout(null):** Utilise un layout nul pour positionner les composants manuellement.

- Ajout de composants au panneau :
1. **JLabel**: Utilisé pour afficher des textes (étiquettes).
  2. **JTextField**: Champs de texte pour entrer le nom d'utilisateur.
  3. **JPasswordField**: Champ de texte pour entrer le mot de passe, masque les caractères.
  4. **JButton**: Boutons pour soumettre ou effacer les informations de connexion.
  5. **setBounds(x, y, width, height)**: Définit la position et la taille des composants.
  6. **setFont**: Change la police des étiquettes.
  7. **setBackground**: Change la couleur de fond des boutons.
  8. **addActionListener(this)**: Associe les boutons à l'écouteur d'événements de la classe actuelle.

```

cabinetLabel = new JLabel("Cabinet Medicale Dr.Boumahra", SwingConstants.RIGHT);
cabinetLabel.setBounds(100, 50, 400, 25);
cabinetLabel.setFont(new Font(cabinetLabel.getFont().getName(), Font.BOLD, 21));
panel.add(cabinetLabel);

usernameLabel = new JLabel("Utilisateur", SwingConstants.RIGHT);
usernameLabel.setBounds(100, 140, 140, 25);
usernameLabel.setFont(new Font(usernameLabel.getFont().getName(), Font.BOLD, usernameLabel.getFont().getSize()));
panel.add(usernameLabel);

usernameTextField = new JTextField(20);
usernameTextField.setBounds(270, 140, 165, 25);
panel.add(usernameTextField);

passwordLabel = new JLabel("Mot de passe", SwingConstants.RIGHT);
passwordLabel.setBounds(120, 200, 135, 25);
passwordLabel.setFont(new Font(passwordLabel.getFont().getName(), Font.BOLD, usernameLabel.getFont().getSize()));
panel.add(passwordLabel);

passwordTextField = new JPasswordField(20);
passwordTextField.setBounds(270, 200, 165, 25);
panel.add(passwordTextField);

loginButton = new JButton("Connexion");
loginButton.addActionListener(this);
loginButton.setBounds(200, 260, 100, 25);
loginButton.setBackground(new Color(22, 189, 179));
panel.add(loginButton);

clearButton = new JButton("Effacer");
clearButton.addActionListener(this);
clearButton.setBounds(310, 260, 80, 25);
clearButton.setBackground(new Color(22, 189, 179));
panel.add(clearButton);

successLabel = new JLabel("Success", SwingConstants.CENTER);
successLabel.setBounds(100, 310, 400, 25);
successLabel.setFont(new Font(successLabel.getFont().getName(), Font.BOLD, usernameLabel.getFont().getSize()));
panel.add(successLabel);
successLabel.setText(null);

```

- Ajout d'une image de fond : (traiter comme une exception )
  1. **BufferedImage image:** Lit une image à partir d'un fichier.
  2. **magelIO.read:** Charge l'image depuis le chemin spécifié.
  3. **JLabel backgroundLabel:** Utilisé pour afficher l'image en tant que fond.
  4. **try-catch:** Gestion des exceptions si le fichier image ne peut pas être lu.

```
BufferedImage image = null;
try {
    image = ImageIO.read(new File("C:/Users/AZUR/Documents/poo.png"));
    backgroundLabel = new JLabel(new ImageIcon(image));
    backgroundLabel.setBounds(0, 0, 736, 1104);
    panel.add(backgroundLabel);
} catch (IOException e) {
    e.printStackTrace();
}
```

L'image :



- Configuration de la deuxième fenêtre (**secondFrame**) :
  1. **frame.setVisible(true)**: Affiche la fenêtre de connexion.
  2. **frame.setResizable(false)**: Empêche le redimensionnement de la fenêtre.
  3. **secondFrame**: Deuxième fenêtre pour les fonctionnalités après la connexion.
  4. **secondFrame.setSize(736, 1104)**: Définit la taille de la deuxième fenêtre.
  5. **secondFrame.setDefaultCloseOperation(JFrame.HIDE\_ON\_CLOSE)**: Cache la fenêtre au lieu de fermer l'application.
  6. **secondPanel**: Panneau pour organiser les composants de la deuxième fenêtre.

```

frame.setVisible(true);
frame.setResizable(false);
// Initialize the second window
secondFrame = new JFrame("Main Cabinet");
secondFrame.setSize(736, 1104);
secondFrame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
JPanel secondPanel = new JPanel();
secondPanel.setLayout(null);
secondFrame.add(secondPanel);
secondFrame.setResizable(false);

```

- Ajout de boutons à la deuxième fenêtre :
  1. **JButton**: Création de boutons pour les différentes sections (Patients, Rendez Vous, Fiches Patients).
  2. **addActionListener**: Ajoute un écouteur d'événements pour chaque bouton, ouvrant les différentes fenêtres.
  3. **setBounds et setBackground**: Définissent la position, la taille et la couleur de fond des boutons.
  4. **secondPanel.setBackground**: Définit la couleur de fond du panneau.
  5. **secondFrame.setVisible(false)**: La deuxième fenêtre est initialement cachée.

```

// Button for Patients
JButton patientsButton = new JButton("Patients");
patientsButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        openPatientsFrame();
    }
});
patientsButton.setBounds(260, 200, 200, 50);
patientsButton.setBackground(new Color(22, 189, 179));
secondPanel.add(patientsButton);

// Button for Rendez Vous
JButton rendezVousButton = new JButton("Rendez Vous");
rendezVousButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        openRendezVousFrame();
    }
});
rendezVousButton.setBounds(260, 300, 200, 50);
rendezVousButton.setBackground(new Color(22, 189, 179));
secondPanel.add(rendezVousButton);

// Set background color for the second window
secondPanel.setBackground(new Color(182, 238, 237));

```

```

// Button for fiches patients
JButton fichespaitentsButton = new JButton("Fiches Patients");
fichespaitentsButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        openfichespaitentsFrame();
    }
});
fichespaitentsButton.setBounds(260, 400, 200, 50);
fichespaitentsButton.setBackground(new Color(22, 189, 179));
secondPanel.add(fichespaitentsButton);

secondFrame.setVisible(false); // Initially hide the second frame

```

## La classe LoginGUI :

### 3. Méthode actionPerformed :

Gère les actions des boutons de connexion et d'effacement :

1. **actionPerformed**: Méthode appelée lorsqu'un bouton est cliqué.

2. **getSource()**: Détermine quel bouton a été cliqué.

3. **loginButton**: Vérifie si le nom d'utilisateur et le mot de passe sont corrects. Si oui, affiche la deuxième fenêtre.

4. **clearButton**: Réinitialise les champs de texte et le message de succès.

5. **setText("")**: Réinitialise le texte des champs et du label.

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == loginButton) {
        String username = usernameTextField.getText();
        char[] passwordChars = passwordTextField.getPassword();
        String password = new String(passwordChars);

        if (users.containsKey(username) && users.get(username).equals(password)) {
            successLabel.setText("connecté !");
            secondFrame.setVisible(true);
            usernameTextField.setText("");
            passwordTextField.setText("");
            successLabel.setText("");
        } else {
            successLabel.setText("Utilisateur ou mot de passe invalide. Veuillez reessayer.");
        }
    } else if (e.getSource() == clearButton) {
        usernameTextField.setText("");
        passwordTextField.setText("");
        successLabel.setText("");
    }
}
```

## La classe LoginGUI :

### **4. Méthode openfichespaitentsFrame :**

Cette méthode est responsable de la création et de la configuration de la fenêtre pour afficher les fiches des patients.

- Création et Configuration de la Fenêtre :

1. **JFrame fichespaitentsFrame** : Crée une nouvelle fenêtre avec le titre "Fiche Patient".

2. **Taille et fermeture** : Définit la taille de la fenêtre et la ferme lorsqu'on clique sur la croix.

3. **JPanel fichespaitentsPanel** : Un panneau pour contenir tous les composants, avec une couleur de fond spécifique et une disposition absolue (positionnement manuel).

4. Ajout du panneau à la fenêtre et interdiction de redimensionnement.

```
private void openfichespaitentsFrame() {  
    JFrame fichespaitentsFrame = new JFrame("Fiche Patient");  
    fichespaitentsFrame.setSize(736, 1104);  
    fichespaitentsFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);  
    JPanel fichespaitentsPanel = new JPanel();  
    fichespaitentsPanel.setBackground(new Color(182, 238, 237));  
    fichespaitentsPanel.setLayout(null); // Set layout to null for absolute positioning  
    fichespaitentsFrame.add(fichespaitentsPanel);  
    fichespaitentsFrame.setResizable(false);
```

- Ajout des Composants : Champs de Texte et Labels :

1. **Labels et champs de texte** : Crée des labels et des champs de texte pour saisir le nom et le prénom du patient. Chaque composant est positionné explicitement avec **setBounds**.

```

// Labels and text fields for patient's name and last name
JLabel nomLabel = new JLabel("Nom:");
nomLabel.setBounds(50, 50, 100, 25);
fichespatientsPanel.add(nomLabel);

JTextField nomTextField = new JTextField();
nomTextField.setBounds(150, 50, 200, 25);
fichespatientsPanel.add(nomTextField);

JLabel prenomLabel = new JLabel("Prenom:");
prenomLabel.setBounds(50, 100, 100, 25);
fichespatientsPanel.add(prenomLabel);

JTextField prenomTextField = new JTextField();
prenomTextField.setBounds(150, 100, 200, 25);
fichespatientsPanel.add(prenomTextField);

```

- Zone de Texte pour les Informations du Patient :
  1. **TextArea** : Utilisé pour afficher les informations du patient. Il est non éditable et formaté pour avoir une police en gras et une taille de police plus grande.
  2. **JScrollPane** : Permet de faire défiler le contenu de la zone de texte si celui-ci dépasse la taille visible.

```

// Text area to display patient information with line wrapping
JTextArea patientInfoTextArea = new JTextArea();
patientInfoTextArea.setEditable(false); // Make it read-only
patientInfoTextArea.setFont(new Font(patientInfoTextArea.getFont().getName(), Font.BOLD, 16));
patientInfoTextArea.setLineWrap(true); // Enable line wrapping
patientInfoTextArea.setWrapStyleWord(true); // Wrap at word boundaries
JScrollPane scrollPane = new JScrollPane(patientInfoTextArea);
scrollPane.setBounds(50, 185, 600, 570); // Set bounds for the panel
fichespatientsPanel.add(scrollPane);

```

## ---> les boutons :

- **Bouton "Chercher":**

Bouton "**Chercher**" : Lance une recherche dans la base de données en utilisant le nom et le prénom saisis par l'utilisateur.

1. **Requête SQL** : Prépare et exécute une requête pour récupérer les informations du patient.
2. **Affichage des résultats** : Si le patient est trouvé, ses informations sont affichées dans la zone de texte.

```
// Button to search for patient information
JButton searchButton = new JButton("Chercher");
searchButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Get the entered patient name and last name
        String nom = nomTextField.getText();
        String prenom = prenomTextField.getText();

        // Retrieve patient information based on Nom and Prenom
        String query = "SELECT * FROM Patient WHERE Nom = ? AND Prenom = ?";
        try {
            PreparedStatement preparedStatement = connection.prepareStatement(query);
            preparedStatement.setString(1, nom);
            preparedStatement.setString(2, prenom);
            ResultSet resultSet = preparedStatement.executeQuery();

            // Display patient information in the text area
            if (resultSet.next()) {
                String patientInfo = "\n \n \n" + "FICHE PATIENT"
                    + "\n \n \n      Nom: " + resultSet.getString("Nom") + "\n"
                    + "\n      Prenom: " + resultSet.getString("Prenom") + "\n"
                    + "\n      Telephone: " + resultSet.getString("Telephone") + "\n"
                    + "\n      Antecedents: " + resultSet.getString("Antecedents") + "\n"
                    + "\n      Observations: " + resultSet.getString("Observations") + "\n"
                    + "\n      Ordonnance: " + resultSet.getString("Ordonnance") + "\n";
                patientInfoTextArea.setText(patientInfo);
            } else {
                patientInfoTextArea.setText("Patient pas trouvÃ©!");
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
            // Handle SQL exception
        }
    }
});
searchButton.setBounds(150, 150, 100, 25);
searchButton.setBackground(new Color(22, 189, 179));
fichespatientsPanel.add(searchButton);
```

- **Bouton "Effacer"** :

1. **Bouton "Effacer"** : Réinitialise les champs de texte et la zone de texte.

```
// Button to clear input fields and text area
JButton clearButton = new JButton("Effacer");
clearButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Clear entered name and last name
        nomTextField.setText("");
        prenomTextField.setText("");
        // Clear patient information text area
        patientInfoTextArea.setText("");
    }
});
clearButton.setBounds(260, 150, 100, 25);
clearButton.setBackground(new Color(22, 189, 179));
fichespatientsPanel.add(clearButton);
```

- **Bouton "imprimer"** :

1. **Bouton "Imprimer"** : Imprime les informations affichées dans la zone de texte en appelant la méthode printText.

```
// Button to print patient information
JButton printButton = new JButton("Imprimer");
printButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String patientInfo = patientInfoTextArea.getText();
        printText(patientInfo);
    }
});
printButton.setBounds(370, 150, 100, 25);
printButton.setBackground(new Color(22, 189, 179));
fichespatientsPanel.add(printButton);
```

- Affichage de la Fenêtre :

1. **Rendre visible la fenêtre** : Affiche la fenêtre à l'écran.

```
fichespatientsFrame.setVisible(true);  
}
```

## --->**La méthode print texte :**

**Cette méthode permet de formater et d'imprimer les informations du patient de manière organisée et professionnelle.**

### **1. Configuration de la Tâche d'Impression :**

- Crée une instance de PrinterJob.
- Définit le nom de la tâche d'impression.
- Configure un objet Printable pour gérer le contenu à imprimer.

### **2. Méthode print :**

- Vérifie si la page demandée est valide.
- Configure le graphique 2D pour l'impression.
- Affiche le nom du médecin et la date en haut de la page.
- Divise le texte à imprimer en lignes pour l'adapter à la zone d'impression.
- Dessine chaque ligne de texte sur la page.

### **3. Affichage des Informations du Patient :**

- Les informations du patient sont récupérées sous forme de texte.
- Ce texte est divisé en lignes pour tenir compte de la taille de la page.
- Chaque ligne est dessinée sur la page d'impression.

```

// Method to print text
private void printText(String text) {
    PrinterJob job = PrinterJob.getPrinterJob();
    job.setJobName("Print Data");

    job.setPrintable(new Printable() {
        public int print(Graphics pg, PageFormat pf, int pageNum) {
            if (pageNum > 0) {
                return Printable.NO_SUCH_PAGE;
            }

            Graphics2D g2 = (Graphics2D) pg;
            g2.translate(pf.getImageableX(), pf.getImageableY());
            g2.setFont(new Font("Arial", Font.PLAIN, 12));

            // Write "Dr.BOUMAHRA" at the top left
            g2.drawString("Dr.BOUMAHRA", 100, 50);

            // Write the date below "Dr.BOUMAHRA"
            // Get the current date and time
            Calendar calendar = Calendar.getInstance();
            SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
            String currentDate = dateFormat.format(calendar.getTime());
            g2.drawString(" Le: " + currentDate, 100, 70);

            // Split the text into lines to fit within the printable area
            String[] lines = text.split("\n");
            int lineHeight = g2.getFontMetrics().getHeight();
            int y = 100; // Initial y position

            // Draw each line of text
            for (String line : lines) {
                g2.drawString(line, 100, y);
                y += lineHeight; // Move to the next line
            }
        }
    });
}

```

```

    // Write "signature et cachet" at the bottom right
    FontMetrics metrics = g2.getFontMetrics();
    String signatureText = "signature et cachet";
    int signatureWidth = metrics.stringWidth(signatureText);
    int pageWidth = (int) pf.getImageableWidth();
    int pageHeight = (int) pf.getImageableHeight();
    int signatureX = pageWidth - signatureWidth - 100;
    int signatureY = pageHeight - 100; // Adjust vertical position
    g2.drawString(signatureText, signatureX, signatureY);

    return Printable.PAGE_EXISTS;
}
});

boolean ok = job.printDialog();
if (ok) {
    try {
        job.print();
    } catch (PrinterException ex) {
        ex.printStackTrace();
    }
}
}

```

## La classe LoginGUI :

### 5. Méthode openpatientsFrame :

1. Initialisation de l'interface graphique:

- créer une **JFrame** pour afficher votre application.
- Un **JPanel** est ajouté à la **JFrame** pour organiser les composants.

```
private void openPatientsFrame() {  
    JFrame patientsFrame = new JFrame("Patients");  
    patientsFrame.setSize(736, 1104);  
    patientsFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);  
    patientsFrame.setResizable(false);  
    JPanel patientsPanel = new JPanel();  
    patientsPanel.setBackground(new Color(182, 238, 237));  
    patientsPanel.setLayout(null); // Set layout to null for absolute positioning  
    patientsFrame.add(patientsPanel);  
}
```

2. Ajout des composants pour les informations des patients:

- on a ajoutée des **JLabels** et **JTextFields** pour saisir les informations des patients, comme le nom, le prénom et le numéro de téléphone.
- on a utilisée également des **JTextAreas** pour les informations supplémentaires telles que les antécédents médicaux, les observations et les ordonnances.

```

// Labels and text fields for patient information
JLabel nameLabel = new JLabel("Nom:");
nameLabel.setBounds(50, 50, 100, 25);
patientsPanel.add(nameLabel);

JTextField nomTextField = new JTextField();
nomTextField.setBounds(150, 50, 200, 25);
patientsPanel.add(nomTextField);

JLabel lastNameLabel = new JLabel("Prenom:");
lastNameLabel.setBounds(50, 100, 100, 25);
patientsPanel.add(lastNameLabel);

JTextField prenomTextField = new JTextField();
prenomTextField.setBounds(150, 100, 200, 25);
patientsPanel.add(prenomTextField);

JLabel telLabel = new JLabel("Telephone:");
telLabel.setBounds(50, 150, 100, 25);
patientsPanel.add(telLabel);

JTextField telephoneTextField = new JTextField();
telephoneTextField.setBounds(150, 150, 200, 25);
patientsPanel.add(telephoneTextField);

// Additional patient information
JLabel antecedentsLabel = new JLabel("Antécédents:");
antecedentsLabel.setBounds(50, 200, 100, 25);
patientsPanel.add(antecedentsLabel);

JTextArea antecedentsTextArea = new JTextArea();
antecedentsTextArea.setLineWrap(true); // Enable line wrapping
antecedentsTextArea.setBounds(150, 200, 500, 60);
patientsPanel.add(antecedentsTextArea);

```

```

JLabel observationsLabel = new JLabel("Observations:");
observationsLabel.setBounds(50, 270, 100, 25);
patientsPanel.add(observationsLabel);

JTextArea observationsTextArea = new JTextArea();
observationsTextArea.setLineWrap(true); // Enable line wrapping
observationsTextArea.setBounds(150, 270, 500, 60);
patientsPanel.add(observationsTextArea);

JLabel ordonnanceLabel = new JLabel("Ordonnance:");
ordonnanceLabel.setBounds(50, 340, 100, 25);
patientsPanel.add(ordonnanceLabel);

JTextArea ordonnanceTextArea = new JTextArea();
ordonnanceTextArea.setLineWrap(true); // Enable line wrapping
ordonnanceTextArea.setBounds(150, 340, 500, 60);
patientsPanel.add(ordonnanceTextArea);

```

## --->Les boutons :

- **Boutons “Sauvgarder” :**

Ce bouton permet à l'utilisateur de sauvegarder les informations d'un nouveau patient.

```
// Button for saving patient information
JButton saveButton = new JButton("Sauvgarder");
saveButton.setBounds(300, 410, 100, 30);
saveButton.setBackground(new Color(22, 189, 179));
patientsPanel.add(saveButton);
```

- **Boutons “Supprimer” :**

Ce bouton permet à l'utilisateur de supprimer le patient sélectionné.

```
// Button for deleting selected patient
JButton deleteButton = new JButton("Supprimer");
deleteButton.setBounds(150, 410, 100, 30);
deleteButton.setBackground(new Color(22, 189, 179));
patientsPanel.add(deleteButton);
```

- **Boutons “Modifier” :**

Ce bouton permet à l'utilisateur de modifier les informations du patient sélectionné.

```
// Button for editing selected patient
JButton editButton = new JButton("Modifier");
editButton.setBounds(450, 410, 100, 30);
editButton.setBackground(new Color(22, 189, 179));
patientsPanel.add(editButton);
```

## --->Création de la table des patients :

### 1. Définir les noms des colonnes :

On définit les noms des colonnes qui seront affichées dans la table des patients

```
// Table  
String[] columnNames = {"Nom", "Prenom", "Telephone", "Antecedents", "Observations", "Ordonnance"};
```

### 2. Créer le modèle de table :

- **DefaultTableModel** est utilisé pour créer le modèle de table avec les colonnes définies.
- **setColumnIdentifiers** permet de spécifier les noms des colonnes.

```
DefaultTableModel model = new DefaultTableModel();  
model.setColumnIdentifiers(columnNames);
```

### • Créer et configurer la JTable :

Une nouvelle instance de **JTable** est créée et le modèle de table est assigné à cette instance.

```
JTable patientTable = new JTable();  
patientTable.setModel(model);
```

### • Ajouter la JTable à un JScrollPane :

1. **JScrollPane** est utilisé pour permettre le défilement de la table si le nombre de lignes dépasse l'espace visible.

2. **setBounds** définit la position et la taille du JScrollPane.

```
JScrollPane scrollPane = new JScrollPane(patientTable);  
scrollPane.setBounds(50, 470, 630, 300);
```

### • Ajouter le JScrollPane au panneau des patients :

Le **JScrollPane** est ajouté au panneau patientsPanel pour qu'il soit visible dans l'interface utilisateur.

```
patientsPanel.add(scrollPane);
```

## --->Chargement des patients existants dans la table:

1. Définir la requête SQL pour sélectionner tous les patients :

On définit une requête SQL simple qui sélectionne toutes les colonnes de la table Patient. Cela nous permet de récupérer toutes les informations des patients existants dans la base de données.

```
// Load existing patients into the table
String query = "SELECT * FROM Patient";
```

2. Exécuter la requête SQL et traiter les résultats :

- **Préparer la requête :** **PreparedStatement** est utilisé pour préparer la requête **SQL**.
- **Exécuter la requête :** **executeQuery** exécute la requête et retourne un **ResultSet** contenant les résultats de la requête.
- **Traiter les résultats :** Le **ResultSet** est parcouru ligne par ligne avec **while (resultSet.next())**. Pour chaque ligne, les données sont récupérées et ajoutées au modèle de table (**model.addRow**).

```
try {
    PreparedStatement preparedStatement = connection.prepareStatement(query);
    ResultSet resultSet = preparedStatement.executeQuery();
    while (resultSet.next()) {
        String nom = resultSet.getString("Nom");
        String prenom = resultSet.getString("Prenom");
        String telephone = resultSet.getString("Telephone");
        String antecedents = resultSet.getString("Antecedents");
        String observations = resultSet.getString("Observations");
        String ordonnance = resultSet.getString("Ordonnance");
        model.addRow(new Object[]{nom, prenom, telephone, antecedents, observations, ordonnance});
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

## --->ActionListener pour le bouton de sauvegarde:

1. Ajouter un ActionListener au bouton de sauvegarde :

On ajoute un **ActionListener** au bouton de sauvegarde (**saveButton**). Cet écouteur attend que le bouton soit cliqué pour exécuter le code à l'intérieur.

```
// Save button action listener  
saveButton.addActionListener(e -> {
```

2. Récupérer les données des champs de texte :

On récupère les données saisies par l'utilisateur dans les différents champs de texte et zones de texte (**nomTextField**, **prenomTextField**, **telephoneTextField**, **antecedentsTextArea**, **observationsTextArea**, **ordonnanceTextArea**).

```
// Retrieve data from text fields  
String nom = nomTextField.getText();  
String prenom = prenomTextField.getText();  
String telephone = telephoneTextField.getText();  
String antecedents = antecedentsTextArea.getText();  
String observations = observationsTextArea.getText();  
String ordonnance = ordonnanceTextArea.getText();
```

2. Définir et préparer la requête SQL d'insertion :

- **Définir la requête d'insertion :** La requête SQL `INSERT INTO Patient (Nom, Prenom, Telephone, Antecedents, Observations, Ordonnance) VALUES (?, ?, ?, ?, ?, ?)` permet d'insérer un nouveau patient dans la base de données.
- **Préparer la requête :** **PreparedStatement** est utilisé pour préparer la requête d'insertion et y insérer les valeurs récupérées des champs de texte.

```
// Insert new patient into the database  
String insertQuery = "INSERT INTO Patient (Nom, Prenom, Telephone, Antecedents, Observations, Ordonnance) VALUES (?, ?, ?, ?, ?, ?)"  
try {  
    PreparedStatement insertStatement = connection.prepareStatement(insertQuery);  
    insertStatement.setString(1, nom);  
    insertStatement.setString(2, prenom);  
    insertStatement.setString(3, telephone);  
    insertStatement.setString(4, antecedents);  
    insertStatement.setString(5, observations);  
    insertStatement.setString(6, ordonnance);  
}
```

### 3.Exécuter la requête d'insertion et mettre à jour le modèle de table :

- **Exécuter la requête : `executeUpdate`** exécute la requête d'insertion. Le nombre de lignes insérées est retourné.
- **Vérifier le succès de l'insertion :** Si une ou plusieurs lignes ont été insérées (**`rowsInserted > 0`**), on ajoute une nouvelle ligne dans le modèle de table (**`model.addRow`**) et on affiche un message de succès avec **JOptionPane**. Sinon, on affiche un message d'échec.
- **Gérer les exceptions :** En cas de problème, l'exception **SQLException** est attrapée et imprimée

```
int rowsInserted = insertStatement.executeUpdate();
if (rowsInserted > 0) {
    model.addRow(new Object[]{nom, prenom, telephone, antecedents, observations, ordonnance});
    JOptionPane.showMessageDialog(saveButton, "Patient ajoute avec succes!");
} else {
    JOptionPane.showMessageDialog(saveButton, "Echec de l'ajout du patient!");
}
} catch (SQLException ex) {
    ex.printStackTrace();
    // Handle SQL exception
}
```

### 4.Effacer les champs de texte après la sauvegarde :

Les champs de texte et les zones de texte sont réinitialisés à des chaînes vides après l'ajout du patient pour préparer l'interface pour une nouvelle entrée.

```
// Clear text fields after saving
nomTextField.setText("");
prenomTextField.setText("");
telephoneTextField.setText("");
antecedentsTextArea.setText("");
observationsTextArea.setText("");
ordonnanceTextArea.setText("");
});
```

## --->ActionListener pour le bouton de suppression :

1.Ajouter un ActionListener au bouton de suppression :

Un **ActionListener** est ajouté au bouton de suppression (**deleteButton**). Cet écouteur attend que le bouton soit cliqué pour exécuter le code à l'intérieur. Il vérifie si une ligne de la table a été sélectionnée (**selectedRow != -1**).

```
// Delete button action listener
deleteButton.addActionListener(e -> {
    int selectedRow = patientTable.getSelectedRow();
    if (selectedRow != -1) {
```

2.Récupérer les valeurs du patient sélectionné :

On récupère les valeurs des colonnes Nom et Prenom de la ligne sélectionnée dans la table.

```
String nom = (String) model.getValueAt(selectedRow, 0);
String prenom = (String) model.getValueAt(selectedRow, 1);
```

3.Définir et préparer la requête SQL de suppression :

- **Définir la requête de suppression :** La requête **SQL DELETE FROM Patient WHERE Nom = ? AND Prenom = ?** permet de supprimer un patient basé sur son nom et prénom.
- **Préparer la requête :** **PreparedStatement** est utilisé pour préparer la requête de suppression et y insérer les valeurs récupérées.

```
String deleteQuery = "DELETE FROM Patient WHERE Nom = ? AND Prenom = ?";
try {
    PreparedStatement deleteStatement = connection.prepareStatement(deleteQuery);
    deleteStatement.setString(1, nom);
    deleteStatement.setString(2, prenom);
```

## 4.Exécuter la requête de suppression et mettre à jour le modèle de table :

- **Exécuter la requête :** **executeUpdate** exécute la requête de suppression. Le nombre de lignes supprimées est retourné.
- **Vérifier le succès de la suppression :** Si une ou plusieurs lignes ont été supprimées (**rowsDeleted > 0**), on supprime la ligne correspondante du modèle de table (**model.removeRow**) et on affiche un message de succès avec **JOptionPane**. Sinon, on affiche un message d'échec.
- **Gérer les exceptions :** En cas de problème, l'exception **SQLException** est attrapée et imprimée.

```
int rowsDeleted = deleteStatement.executeUpdate();
if (rowsDeleted > 0) {
    model.removeRow(selectedRow);
    JOptionPane.showMessageDialog(deleteButton, "Patient supprimé avec succès !");
} else {
    JOptionPane.showMessageDialog(deleteButton, "Échec de la suppression du patient!");
}
} catch (SQLException ex) {
    ex.printStackTrace();
    // Handle SQL exception
}
```

## 5.Effacer les champs de texte après la suppression :

Les champs de texte et les zones de texte sont réinitialisés à des chaînes vides après la suppression du patient pour préparer l'interface pour une nouvelle entrée ou modification. Si aucune ligne n'est sélectionnée, un message demandant à l'utilisateur de sélectionner un patient est affiché avec **JOptionPane**.

```
// Clear input fields after updating
nomTextField.setText("");
prenomTextField.setText("");
telephoneTextField.setText("");
antecedentsTextArea.setText("");
observationsTextArea.setText("");
ordonnanceTextArea.setText("");

} else {
    JOptionPane.showMessageDialog(deleteButton, "Veuillez sélectionner un patient à supprimer !");
}
});
```

## --->ListSelectionListener pour le tableau :

### 1.Ajouter un **ListSelectionListener** à la **JTable** :

Un ListSelectionListener est ajouté au modèle de sélection de la table. Cet écouteur est déclenché lorsque la sélection change. Il vérifie si la sélection n'est pas en cours d'ajustement (**!e.getValueIsAdjusting()**) et si une ligne est sélectionnée (**selectedRow != -1**).

```
// Add a ListSelectionListener to the JTable
patientTable.getSelectionModel().addListSelectionListener(e -> {
    if (!e.getValueIsAdjusting()) {
        int selectedRow = patientTable.getSelectedRow();
        if (selectedRow != -1) {
```

### 2.Remplir les champs de texte avec les informations du patient sélectionné :

Lorsque l'utilisateur sélectionne une ligne dans la table, les informations de cette ligne sont récupérées et affichées dans les champs de texte et les zones de texte correspondants. Cela permet de visualiser et de modifier les informations du patient sélectionné.

```
// Populate the input fields with the selected patient's information
nomTextField.setText((String) model.getValueAt(selectedRow, 0));
prenomTextField.setText((String) model.getValueAt(selectedRow, 1));
telephoneTextField.setText((String) model.getValueAt(selectedRow, 2));
antecedentsTextArea.setText((String) model.getValueAt(selectedRow, 3));
observationsTextArea.setText((String) model.getValueAt(selectedRow, 4));
ordonnanceTextArea.setText((String) model.getValueAt(selectedRow, 5));
```

## --->ActionListener pour le bouton de modification :

### 1.Ajouter un ActionListener au bouton de modification :

Un **ActionListener** est ajouté au bouton de modification (**editButton**). Cet écouteur attend que le bouton soit cliqué pour exécuter le code à l'intérieur. Il vérifie si une ligne de la table a été sélectionnée (**selectedRow != -1**).

```
// Edit button action listener
editButton.addActionListener(e -> {
    int selectedRow = patientTable.getSelectedRow();
    if (selectedRow != -1) {
```

## 2.Récupérer les nouvelles informations saisies dans les champs de texte:

On récupère les nouvelles données saisies par l'utilisateur dans les différents champs de texte et zones de texte.

```
// Get the selected patient's information from input fields
String nom = nomTextField.getText();
String prenom = prenomTextField.getText();
String telephone = telephoneTextField.getText();
String antecedents = antecedentsTextArea.getText();
String observations = observationsTextArea.getText();
String ordonnance = ordonnanceTextArea.getText();
```

## 3.Mettre à jour les informations du patient dans le modèle de table :

Les nouvelles informations du patient sont mises à jour dans le modèle de table. Cela met à jour l'affichage de la table avec les nouvelles valeurs.

```
// Update the selected patient's information in the table and the database
model.setValueAt(nom, selectedRow, 0);
model.setValueAt(prenom, selectedRow, 1);
model.setValueAt(telephone, selectedRow, 2);
model.setValueAt(antecedents, selectedRow, 3);
model.setValueAt(observations, selectedRow, 4);
model.setValueAt(ordonnance, selectedRow, 5);
```

## 4. Définir et préparer la requête SQL de mise à jour :

- Définir la requête de mise à jour :** La requête SQL **UPDATE Patient SET Nom=? , Prenom=? , Telephone=? , Antecedents=? , Observations=? , Ordonnance=? WHERE Nom=? AND Prenom=?** permet de mettre à jour un patient basé sur son nom et prénom précédents.
- Préparer la requête : PreparedStatement** est utilisé pour préparer la requête de mise à jour et y insérer les nouvelles valeurs ainsi que les anciennes valeurs de nom et prénom.

```
// Implement logic to update the patient's information in the database
String updateQuery = "UPDATE Patient SET Nom=?, Prenom=?, Telephone=?, Antecedents=?, Observations=?, Ordonnance=? WHERE Nom=? AND Prenom=?";
try {
    PreparedStatement updateStatement = connection.prepareStatement(updateQuery);
    updateStatement.setString(1, nom);
    updateStatement.setString(2, prenom);
    updateStatement.setString(3, telephone);
    updateStatement.setString(4, antecedents);
    updateStatement.setString(5, observations);
    updateStatement.setString(6, ordonnance);
    updateStatement.setString(7, (String) model.getValueAt(selectedRow, 0)); // Previous Nom
    updateStatement.setString(8, (String) model.getValueAt(selectedRow, 1)); // Previous Prenom
```

```
// Edit button action listener
editButton.addActionListener(e -> {
    int selectedRow = patientTable.getSelectedRow();
    if (selectedRow != -1) {
```

## 2.Récupérer les nouvelles informations saisies dans les champs de texte:

On récupère les nouvelles données saisies par l'utilisateur dans les différents champs de texte et zones de texte.

```
// Get the selected patient's information from input fields
String nom = nomTextField.getText();
String prenom = prenomTextField.getText();
String telephone = telephoneTextField.getText();
String antecedents = antecedentsTextArea.getText();
String observations = observationsTextArea.getText();
String ordonnance = ordonnanceTextArea.getText();
```

## 3.Mettre à jour les informations du patient dans le modèle de table :

Les nouvelles informations du patient sont mises à jour dans le modèle de table. Cela met à jour l'affichage de la table avec les nouvelles valeurs.

```
// Update the selected patient's information in the table and the database
model.setValueAt(nom, selectedRow, 0);
model.setValueAt(prenom, selectedRow, 1);
model.setValueAt(telephone, selectedRow, 2);
model.setValueAt(antecedents, selectedRow, 3);
model.setValueAt(observations, selectedRow, 4);
model.setValueAt(ordonnance, selectedRow, 5);
```

## 4. Définir et préparer la requête SQL de mise à jour :

- Définir la requête de mise à jour :** La requête SQL **UPDATE Patient SET Nom=? , Prenom=? , Telephone=? , Antecedents=? , Observations=? , Ordonnance=? WHERE Nom=? AND Prenom=?** permet de mettre à jour un patient basé sur son nom et prénom précédents.
- Préparer la requête : PreparedStatement** est utilisé pour préparer la requête de mise à jour et y insérer les nouvelles valeurs ainsi que les anciennes valeurs de nom et prénom.

```
// Implement logic to update the patient's information in the database
String updateQuery = "UPDATE Patient SET Nom=?, Prenom=?, Telephone=?, Antecedents=?, Observations=?, Ordonnance=? WHERE Nom=? AND Prenom=?";
try {
    PreparedStatement updateStatement = connection.prepareStatement(updateQuery);
    updateStatement.setString(1, nom);
    updateStatement.setString(2, prenom);
    updateStatement.setString(3, telephone);
    updateStatement.setString(4, antecedents);
    updateStatement.setString(5, observations);
    updateStatement.setString(6, ordonnance);
    updateStatement.setString(7, (String) model.getValueAt(selectedRow, 0)); // Previous Nom
    updateStatement.setString(8, (String) model.getValueAt(selectedRow, 1)); // Previous Prenom
```

## 5.Exécuter la requête de mise à jour et gérer les résultats :

- **Exécuter la requête** : executeUpdate exécute la requête de mise à jour. Le nombre de lignes mises à jour est retourné.
- **Vérifier le succès de la mise à jour** : Si une ou plusieurs lignes ont été mises à jour (**rowsUpdated > 0**), un message de succès est affiché avec **JOptionPane**. Sinon, un message d'échec est affiché.
- **Gérer les exceptions** : En cas de problème, l'exception **SQLException** est attrapée et imprimée.

```
int rowsUpdated = updateStatement.executeUpdate();
if (rowsUpdated > 0) {
    JOptionPane.showMessageDialog(editButton, "Patient modifié avec succès !");
} else {
    JOptionPane.showMessageDialog(editButton, "Échec de la modification du patient!");
}
} catch (SQLException ex) {
    ex.printStackTrace();
    // Handle SQL exception
}
```

## 6.Effacer les champs de texte après la mise à jour :

Les champs de texte et les zones de texte sont réinitialisés à des chaînes vides après la mise à jour du patient pour préparer l'interface pour une nouvelle entrée ou modification. Si aucune ligne n'est sélectionnée, un message demandant à l'utilisateur de sélectionner un patient est affiché avec **JOptionPane**.

```
// Clear input fields after updating
nomTextField.setText("");
prenomTextField.setText("");
telephoneTextField.setText("");
antecedentsTextArea.setText("");
observationsTextArea.setText("");
ordonnanceTextArea.setText("");
} else {
    JOptionPane.showMessageDialog(editButton, "Veuillez sélectionner un patient à supprimer !");
}
});
```

## La classe LoginGUI :

### 6. Méthode openrendezvousFrame :

- Création de la fenêtre JFrame et du panneau JPanel :
- Cette partie crée une fenêtre (**JFrame**) avec le titre "Rendez Vous".
- L'opération de fermeture est définie sur **DISPOSE\_ON\_CLOSE**, ce qui signifie que la fenêtre sera simplement cachée lorsqu'elle est fermée.
- La fenêtre n'est pas redimensionnable (**setResizable(false)**) pour conserver la mise en page.
- Un panneau (**JPanel**) est ajouté à la fenêtre pour organiser les composants graphiques.

```
private void openRendezVousFrame() {  
    JFrame rendezVousFrame = new JFrame("Rendez Vous");  
    rendezVousFrame.setSize(736, 1104);  
    rendezVousFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);  
    rendezVousFrame.setResizable(false);  
    JPanel rendezVousPanel = new JPanel();  
    rendezVousPanel.setBackground(new Color(182, 238, 237));  
    rendezVousPanel.setLayout(null); // Set layout to null for absolute positioning  
    rendezVousFrame.add(rendezVousPanel);
```

- Ajout des composants graphiques :
  1. Des étiquettes (**JLabel**) sont ajoutées pour afficher des textes explicatifs comme "**Nom:**", "**Prenom:**", etc.
  2. Des champs de texte (**JTextField**) sont ajoutés pour permettre à l'utilisateur de saisir le nom et le prénom.
  3. Des menus déroulants (**JComboBox**) sont utilisés pour la sélection de la date et de l'heure.

```

// JLabels
JLabel nomLabel = new JLabel("Nom:");
nomLabel.setBounds(50, 50, 100, 25);
rendezVousPanel.add(nomLabel);

JLabel prenomLabel = new JLabel("Prénom:");
prenomLabel.setBounds(50, 100, 100, 25);
rendezVousPanel.add(prenomLabel);

JLabel dateLabel = new JLabel("Date et Heure:");
dateLabel.setBounds(50, 150, 150, 25);
rendezVousPanel.add(dateLabel);

// JTextFields
JTextField nomTextField = new JTextField();
nomTextField.setBounds(150, 50, 200, 25);
rendezVousPanel.add(nomTextField);

JTextField prenomTextField = new JTextField();
prenomTextField.setBounds(150, 100, 200, 25);
rendezVousPanel.add(prenomTextField);

// Combo boxes for date and hour selection
String[] annee = {"2024", "2025", "2026"}; // Sample years, you can adjust these
JComboBox<String> anneeComboBox = new JComboBox<>(annee);
anneeComboBox.setBounds(150, 150, 100, 25);
rendezVousPanel.add(anneeComboBox);

String[] mois = {"janvier", "fevrier", "mars", "avril", "mai", "juin", "juillet", "aout", "septembre", "octobre", "novembre", "decembre"};
JComboBox<String> moisComboBox = new JComboBox<>(mois);
moisComboBox.setBounds(260, 150, 100, 25);
rendezVousPanel.add(moisComboBox);

String[] jour = new String[31];
for (int i = 0; i < 31; i++) {
    jour[i] = String.valueOf(i + 1);
}

```

```

JComboBox<String> jourComboBox = new JComboBox<>(jour);
jourComboBox.setBounds(370, 150, 100, 25);
rendezVousPanel.add(jourComboBox);

String[] heure = new String[24];
for (int i = 0; i < 24; i++) {
    heure[i] = String.format("%02d", i);
}
JComboBox<String> heureComboBox = new JComboBox<>(heure);
heureComboBox.setBounds(480, 150, 60, 25);
rendezVousPanel.add(heureComboBox);

```

- Bouton "Sauvegarder" (submitButton) :

1. Ce code crée un bouton "**Sauvegarder**" qui exécute une action lorsqu'il est cliqué.
2. L'action consiste à récupérer les données saisies par l'utilisateur (**nom, prénom, date et heure**), à construire une requête SQL d'insertion avec ces données, puis à l'exécuter.
3. Après l'insertion réussie, la méthode **refreshTable()** est appelée pour mettre à jour le tableau avec les nouvelles données.
4. Les champs de texte et les menus déroulants sont effacés pour permettre à l'utilisateur de saisir de nouveaux **rendez-vous**.

```

// Submit Button
JButton submitButton = new JButton("Sauvegarder");
submitButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String nom = nomTextField.getText();
        String prenom = prenomTextField.getText();
        String annee = (String) anneeComboBox.getSelectedItem();
        String mois = (String) moisComboBox.getSelectedItem();
        String jour = (String) jourComboBox.getSelectedItem();
        String heure = (String) heureComboBox.getSelectedItem();
        String query1 = "INSERT INTO rendez_vous VALUES ('" + nom + "','" + prenom + "','" + annee + "','" + mois + "','" + jour + "','" + heure + "')";

        try {
            statement = connection.createStatement();
            statement.executeUpdate(query1);
            JOptionPane.showMessageDialog(submitButton, "Insertion réussie !");
            refreshTable(); // Refresh the table after insertion
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }
});
submitButton.setBounds(150, 200, 110, 30);
submitButton.setBackground(new Color(22, 189, 179)); // Set background color
rendezVousPanel.add(submitButton);

```

- Tableau (JTable):

1. Cette partie crée un tableau avec les colonnes "**Nom**", "**Prenom**", "**Annee**", "**Mois**", "**Jour**" et "**Heure**".
2. Un modèle de tableau par défaut (**DefaultTableModel**) est utilisé pour stocker les données du tableau.
3. Un objet **JScrollPane** est utilisé pour ajouter des barres de défilement si le contenu du tableau dépasse la taille de la fenêtre.

```

// Table
String[] columnNames = {"Nom ", "Prenom", "Annee", "Mois", "Jour", "Heure"};

DefaultTableModel model = new DefaultTableModel();
model.setColumnIdentifiers(columnNames);
table = new JTable();
table.setModel(model);
JScrollPane scrollPane = new JScrollPane(table);
scrollPane.setBounds(50, 250, 600, 400);
rendezVousPanel.add(scrollPane);

// Populate table with existing data
populateTable(model);

```

- Bouton "Effacer" (deleteButton) :

1. Ce code crée un bouton "**Effacer**" qui permet de supprimer un rendez-vous sélectionné dans le tableau.
2. Un écouteur d'événements est ajouté au bouton pour gérer l'action de suppression.

3.Lorsque le bouton est cliqué, il récupère la ligne sélectionnée dans le tableau, extrait les données du rendez-vous (nom, prénom, année, mois, jour, heure) et exécute une requête SQL pour supprimer ce rendez-vous de la base de données.

4.Si la suppression réussit, la ligne correspondante est également retirée du modèle de tableau (DefaultTableModel), et un message de confirmation est affiché. Sinon, un message d'erreur est affiché.

```
// Delete Button
JButton deleteButton = new JButton("Effacer");
deleteButton.setBounds(300, 200, 100, 30);
deleteButton.setBackground(new Color(22, 189, 179));
rendezVousPanel.add(deleteButton);

// Add action listener for the delete button
deleteButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int selectedRow = table.getSelectedRow();
        if (selectedRow != -1) {
            String nom = (String) table.getValueAt(selectedRow, 0);
            String prenom = (String) table.getValueAt(selectedRow, 1);
            String annee = (String) table.getValueAt(selectedRow, 2);
            String mois = (String) table.getValueAt(selectedRow, 3);
            String jour = (String) table.getValueAt(selectedRow, 4);
            String heure = (String) table.getValueAt(selectedRow, 5);

            String deleteQuery = "DELETE FROM rendez_vous WHERE nom = ? AND prenom = ? AND annee = ? AND mois = ? AND jour = ? AND heure = ?";
            try {
                PreparedStatement deleteStatement = connection.prepareStatement(deleteQuery);
                deleteStatement.setString(1, nom);
                deleteStatement.setString(2, prenom);
                deleteStatement.setString(3, annee);
                deleteStatement.setString(4, mois);
                deleteStatement.setString(5, jour);
                deleteStatement.setString(6, heure);
                int rowsDeleted = deleteStatement.executeUpdate();
                if (rowsDeleted > 0) {
                    DefaultTableModel model = (DefaultTableModel) table.getModel();
                    model.removeRow(selectedRow);
                    JOptionPane.showMessageDialog(deleteButton, "Rendez-vous supprimé avec succès!");
                } else {
                    JOptionPane.showMessageDialog(deleteButton, "Echec suppression rendez-vous!");
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
    }
}); // Handle SQL exception
} else {
    JOptionPane.showMessageDialog(deleteButton, "Selectionner un rendez-vous à effacer!");
}
});
rendezVousFrame.setVisible(true);
}
```

- Méthode **populateTable(DefaultTableModel model)** :

- 1.Cette méthode est appelée lors de l'initialisation de l'interface utilisateur pour remplir le tableau avec les données existantes de la base de données.
- 2.Une requête **SQL SELECT** est exécutée pour récupérer tous les rendez-vous de la table **rendez\_vous**.

- 3.Les données récupérées sont ensuite ajoutées au modèle de tableau (**DefaultTableModel**) pour les afficher à l'utilisateur.

```
// Method to populate the table with existing data from the database
private void populateTable(DefaultTableModel model) {
    try {
        String query = "SELECT * FROM rendez_vous";
        statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(query);
        while (resultSet.next()) {
            String nom = resultSet.getString("nom");
            String prenom = resultSet.getString("prenom");
            String annee = resultSet.getString("annee");
            String mois = resultSet.getString("mois");
            String jour = resultSet.getString("jour");
            String heure = resultSet.getString("heure");
            model.addRow(new Object[]{nom, prenom, annee, mois, jour, heure});
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- Méthode **refreshTable()** :

- 1.Cette méthode est utilisée pour rafraîchir le contenu du tableau après l'insertion ou la suppression de données.
- 2.Elle efface d'abord toutes les lignes existantes du modèle de tableau (**DefaultTableModel**) en utilisant **model.setRowCount(0)**.
- 3.Ensuite, elle rappelle la méthode **populateTable()** pour remplir à nouveau le tableau avec les données mises à jour.

```
// Method to refresh the table after data insertion
private void refreshTable() {
    DefaultTableModel model = (DefaultTableModel) table.getModel();
    model.setRowCount(0); // Clear existing data
    populateTable(model); // Repopulate the table with updated data
}
```

## La classe LoginGUI :

### 6. Méthode main :

- Signature de la méthode main :

1. **public static void** : La méthode est publique (accessible depuis n'importe quelle autre classe) et statique (appelée sans avoir besoin d'instancier la classe).

2. **main** : Nom de la méthode. C'est le point d'entrée du notre programme Java.

```
public static void main(String[] args) {
```

- Bloc try-catch et Chargement du pilote JDBC et établissement de la connexion à la base de données Oracle :

**1.try** : Bloc de code où les exceptions sont surveillées.

**2.catch (Exception e1)** : Si une exception est levée dans le bloc try, elle est attrapée ici et traitée. Exception est le type d'exception générique attrapant toutes les exceptions. e1 est une référence à l'objet exception, utilisée pour obtenir des informations sur l'exception.

**3.e1.printStackTrace();** : Cela imprime les détails de l'exception sur la console, utile pour le débogage.

**4.Class.forName("oracle.jdbc.driver.OracleDriver");** : Charge le pilote JDBC Oracle, permettant à Java de communiquer avec la base de données Oracle.

**5.DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "merakchi", "rahma");** : Établit une connexion avec la base de données Oracle spécifiée par l'URL JDBC.

```
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "merakchi", "rahma");
} catch (Exception e1) {
    e1.printStackTrace();
}
```

- Création d'une instance de LoginGUI :

Cela crée une nouvelle instance de notre classe LoginGUI (la calasse globale de tout le projet )

```
'  
new LoginGUI();
```

et apres exécution de tout ce programme notre petit projet est prêt a manipuler et a decouvrirre .

# La base de données

La base de données présentée se compose de deux tables principales : **Patient** et **rendez\_vous**. Voici une description détaillée de chaque table et de leur structure :

## 1 \Table Patient :

Cette table stocke les informations des patients. Voici les colonnes définies :

- **Nom** : Le nom du patient, de type **VARCHAR2** avec une longueur maximale de **100 caractères**.
- **Prenom** : Le prénom du patient, de type **VARCHAR2** avec une longueur maximale de 100 caractères.
- **Telephone** : Le numéro de téléphone du patient, de type **NUMBER** avec une longueur maximale de **10 chiffres**.
- **Antecedents** : Les antécédents médicaux du patient, de type **VARCHAR2** avec une longueur maximale de **100 caractères**.
- **Observations** : Les observations médicales sur le patient, de type **VARCHAR2** avec une longueur **non précise** .
- **Ordonnance** : Les prescriptions médicales du patient, de type **VARCHAR2** avec une longueur **non précise** .

Cette table utilise une clé primaire composite (**PRIMARY KEY**) composée des colonnes **Nom** et **Prenom**, ce qui garantit l'unicité de chaque enregistrement de patient basé sur ces deux colonnes.

```

CREATE TABLE Patient (
    Nom VARCHAR2(100),
    Prenom VARCHAR2(100),
    Telephone number(10),
    Antecedents VARCHAR2,
    Observations VARCHAR2,
    Ordonnance VARCHAR2,
    constraint pk_Patient PRIMARY KEY (Nom,Prenom)
);

```

## 2\Table rendez\_vous :

Cette table stocke les informations sur les **rendez-vous** des patients. Voici les colonnes définies :

- **Nom** : Le nom du patient pour lequel le rendez-vous est fixé, de type **VARCHAR2** avec une longueur maximale de **100 caractères**.
- **Prenom** : Le prénom du patient pour lequel le rendez-vous est fixé, de type **VARCHAR2** avec une longueur maximale **de 100 caractères**.
- **Annee** : L'année du rendez-vous, de type **VARCHAR2** avec une longueur maximale de 100 caractères.
- **Mois** : Le mois du rendez-vous, de type **VARCHAR2** avec une longueur maximale de **100 caractères**.
- **Jour** : Le jour du rendez-vous, de type **VARCHAR2** avec une longueur maximale de **100 caractères**.
- **Heure** : L'heure du rendez-vous, de type **VARCHAR2** avec une longueur maximale de **100 caractères**.

--->Cette table utilise également une clé primaire composite (**PRIMARY KEY**) composée des colonnes **Nom et Prenom**, ce qui garantit l'unicité de chaque rendez-vous basé sur ces deux colonnes.

Remarques

1. Clé primaire composite : Dans les deux tables, la clé primaire est composée des colonnes **Nom et Prenom**. Cela signifie que chaque combinaison de **Nom et Prenom** doit être unique dans chaque table.

2. Types de données : Les types de données utilisés (**VARCHAR2 et NUMBER**) sont appropriés pour stocker des informations textuelles et numériques respectivement

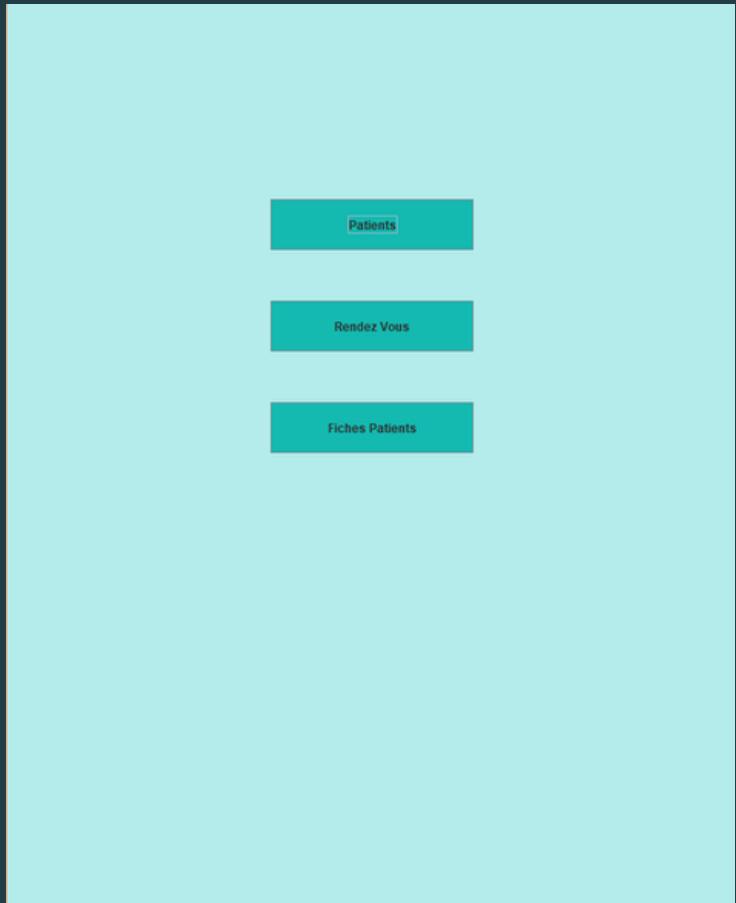
```
CREATE TABLE rendez_vous (
    Nom VARCHAR2(100),
    Prenom VARCHAR2(100),
    Annee VARCHAR2(100),
    Mois VARCHAR2(100),
    Jour VARCHAR2(100),
    Heure VARCHAR2(100),
    constraint pk_rendez_vous PRIMARY key (Nom,Prenom)
);
```

# OF Les interfaces graphiques

## 1\ La première interface :



**Tout d'abord, utilisateur doit se connecter avec un nom ,utilisateur et un mot de passe ( les medecins et les secretaire )**

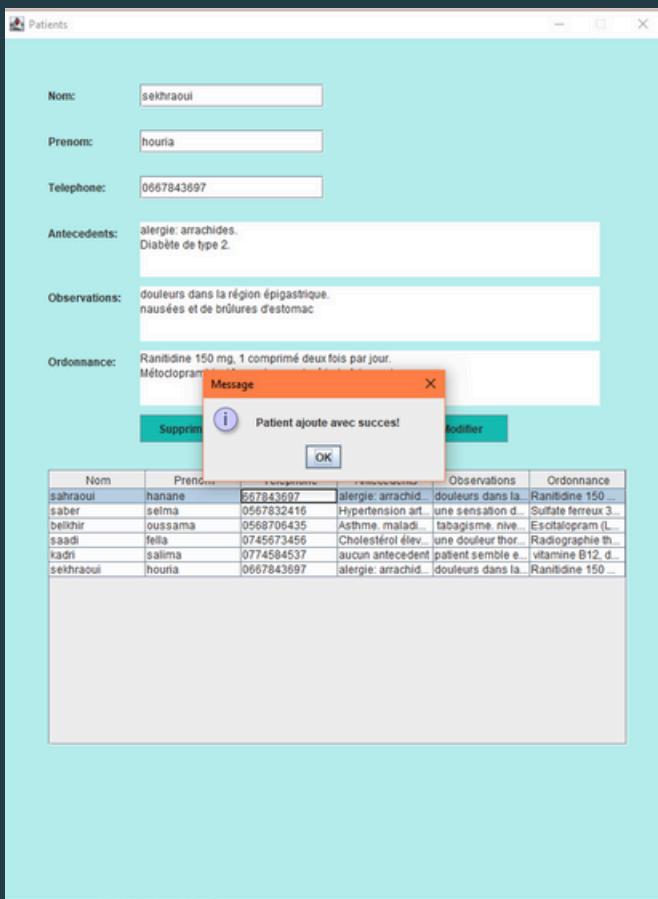


**Si le médecin clique sur le Patients il aura cet affichage :**

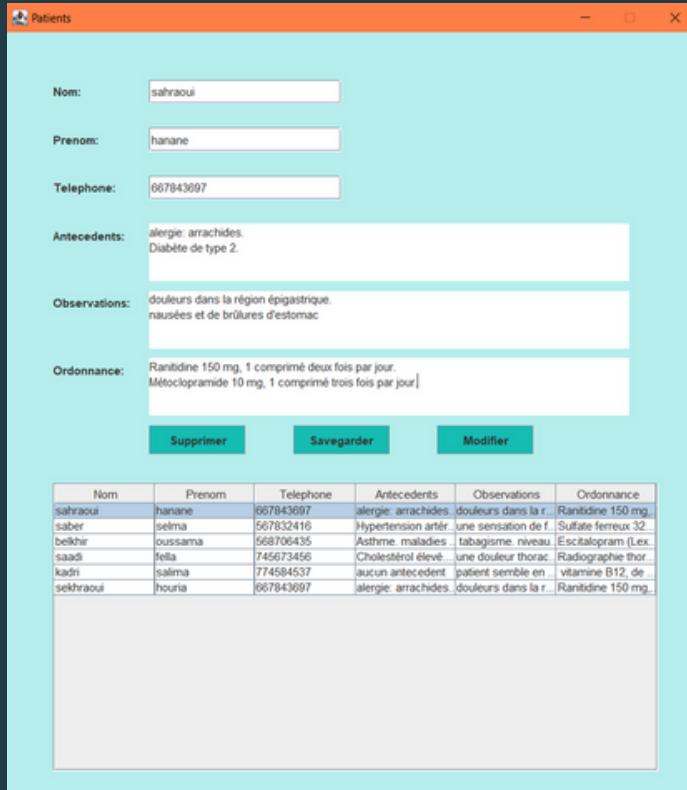
Nom	Prenom	Telephone	Antecedents	Observations	Ordonnance
sahraoui	hanane	667843697	alergie: arrachid...	douleurs dans la...	Ranitidine 150 ...
saber	selma	567832416	Hypertension art...	une sensation d...	Sulfate ferreux 3...
belhmir	oussama	568706435	Asthme malad...	tabagisme, nive...	Esotalopram (L...
saadi	feïla	745673456	Cholestérol élevé...	une douleur thor...	Radiographie th...
kadir	salima	774584537	aucun antécédent	patient semble e...	vitamine B12, d...
sekhraoui	houma	667843697	alergie: arrachid...	douleurs dans la...	Ranitidine 150 ...

**2\ La 2 ème interface :  
Une fois connecté,  
l'utilisateur  
peu utiliser différentes  
fonctionnalités de gestion,  
notamment la gestion des  
patients(les médecins), la  
gestion des dossiers  
patients et la planification  
des rendez-vous(les  
secrétaires).**

**3\ La 3 ème  
interface :  
dans cette  
interface , le  
médecin a la  
possibilité de  
ajouter , supprimer  
ou modifier les  
informations d'un  
patient .**



**Après que le médecin a terminé de saisir les informations, il clique sur le bouton Sauvegarder . un message de confirmation de l'ajout du patient sera affichée**



**Une fois le patient est sauvegardé dans la liste des patients, le médecin peut le sélectionner dans la table et effectuer des modifications sur ses informations ou le supprimer.**

Message

X



Veuillez selectionner un patient a supprimer !

OK

Message

X



Veuillez selectionner un patient a modifier !

OK

Antécédents

OBSERVATIONS

Ordonnance

**dans le cas ou le médecin n'a pas sélectionné un patients de la table est clique sur le bouton modifier ou supprimer un message qui lui attribut cette demande sera afficher**

**dans le cas contraire le patients selectionner et bien modifier ou bien supprimer avec un message de confirmation**

Patients

Nom: saadi

Prenom: fella

Telephone: 745673456

Antecedents: Cholestérol élevé.  
alergie aux graines de pollen

Observations: une douleur thoracique de type serrement depuis environ une heure.

Ordonnance: Radiographie thoracique (vue de face et de profil)

Supprimer

Savegarder

Modifier

Patients

Nom: sahraoui

Prenom: hanane

Telephone: 667843697

Antecedents: alergie: arrachides.  
Diabète de type 2.

Observations: aucune douleurs dans la région épigastrique.  
nausées

Ordonnance: Ranitidine 150 mg, 1 comprimé deux fois par jour.  
Méclizine 10 mg, 1 comprimé trois fois par jour.

Supprimer

Savegarder

Modifier

Nom	Prenom	Telephone	Antecedents	Observations	Ordonnance
sahraoui	hanane	667843697	alergie: arrachides.	aucune douleurs d...	Ranitidine 150 mg...
saber	selma	567832416	Hypertension artér...	une sensation de f...	Sulfate ferreux 32...
belkhir	oussama	568706435	Asthme, maladies...	tabagisme niveau...	Escitalopram (Lex...
kadri	salima	774584537	aucun antecedent...	patient semble en...	vitamine B12, de...
sekhraoui	houria	667843697	alergie: arrachides.	douleurs dans la r...	Ranitidine 150 mg...

i Patient modifié avec succès!

OK

Message

X



Patient supprimé avec succès!

OK

Nom:

Prenom:

Date et Heure: 2024  Janvier  1  00

**Sauvegarder** **Effacer**

Nom	Prenom	Année	Mois	Jour	Heure
hesnaoui	lamia	2024	juin	6	14
saber	selma	2024	mai	11	08
djaoud	karim	2024	juin	16	10
sahli	farah	2024	juillet	15	08
sekhraoui	houria	2024	septembre	11	10
tahraoui	racim	2025	mars	2	08

**4\ La 4 ème interface :**  
**dans cette interface , le secrétaire a la possibilité de d'atribuer un rendez vous e un patient ou bien le effacer .**

Nom: hesnaoui

Prenom: lamia

Date et Heure: 2024  juin  6  14

**Sauvegarder** **Effacer**

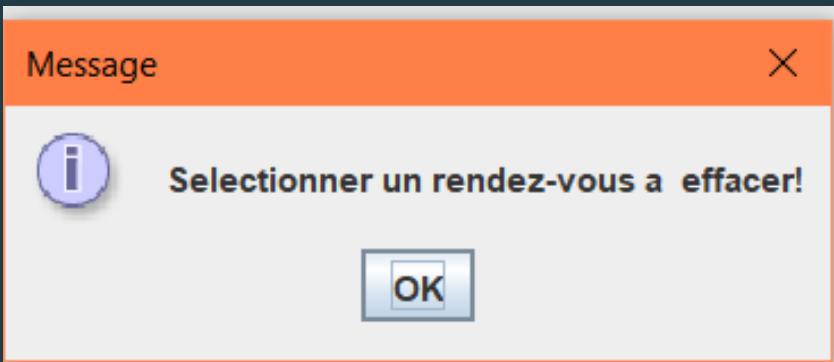
Nom	Prenom	Année	Mois	Jour	Heure
hesnaoui	lamia	2024	juin	6	14

**Message**

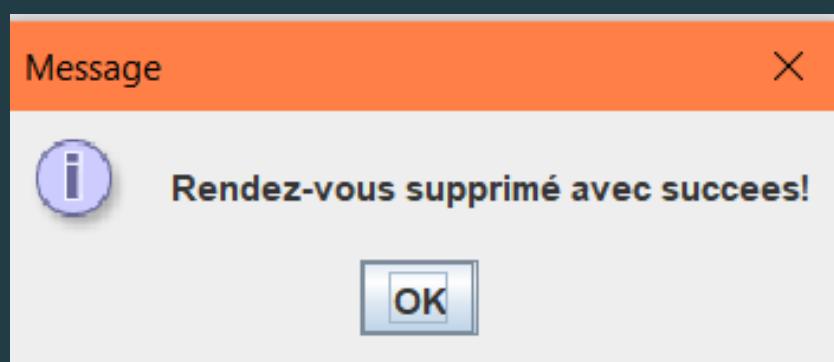
Insertion réussie :)

**OK**

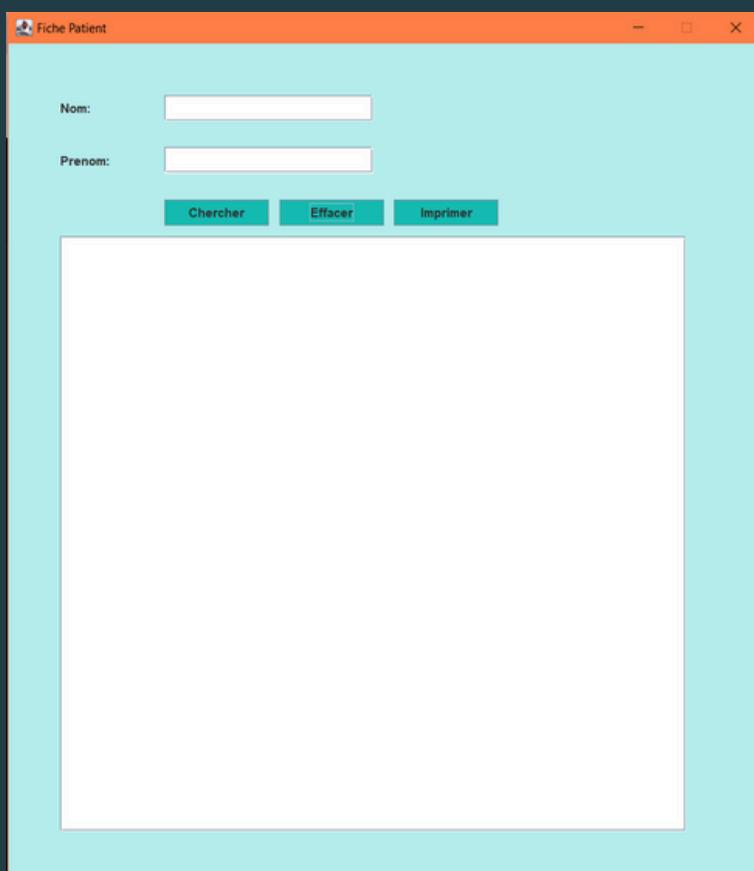
**après la saisie des données par le secrétaire ,le rendez-vous sera insérer avec succès avec un message de confirmation qui s'affiche sur l'écran après avoir cliqué sur le bouton sauvgarder**



**si le secrétaire appuie sur le bouton supprimer sans selectionner un rendez-vous de la table le message a côté sera affiché a l'écran.**



**dans le cas contraire le rendez vous sera supprimé avec succès .**



**5\ La 5ème interface :**  
**dans cette interface , le médecin génère la fiche patient pour chaque patient qui dans le système .il suffit qu'il entre le nom et le prénom de ce dernier.**

**Fiche Patient**

Nom:   
Prenom:

**Chercher** **Effacer** **Imprimer**

**FICHE PATIENT**

- Nom: saber
- Prenom: selma
- Telephone: 567832416
- Antecedents: Hypertension artérielle.  
Des fractures osseuses.
- Observations: une sensation de faiblesse générale .  
une transpiration abondante.
- Ordonnance: Sulfate ferreux 325 mg, 1 comprimé deux fois par jour.  
Ferrous fumarate 200 mg, 1 comprimé trois fois par jour

**dans le cas de la saisie d'un nom et d'un prénom d'un patient qui est dans le système ,une affiche sera affichée sur l'écran du médecin**

**Fiche Patient**

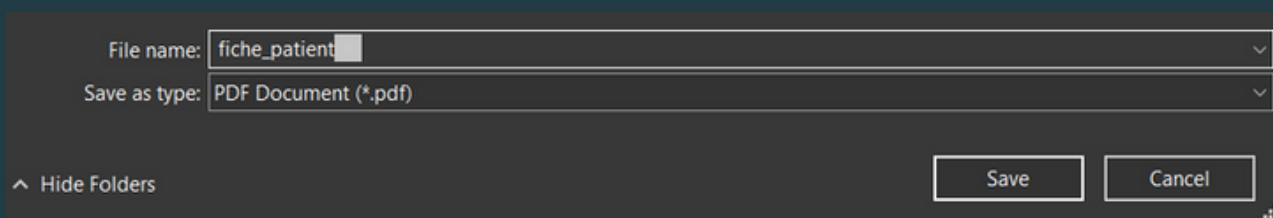
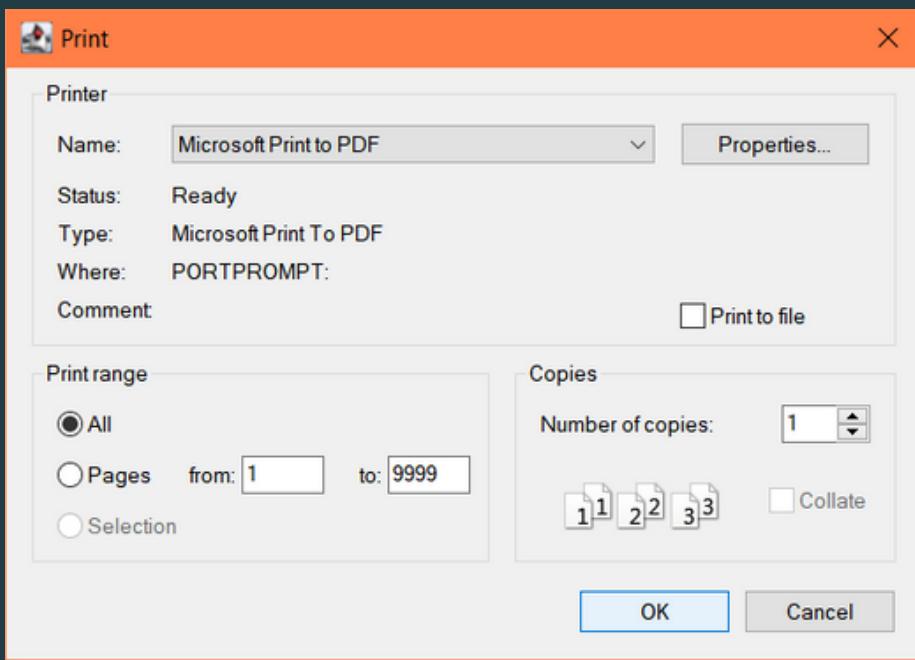
Nom:   
Prenom:

**Chercher** **Effacer** **Imprimer**

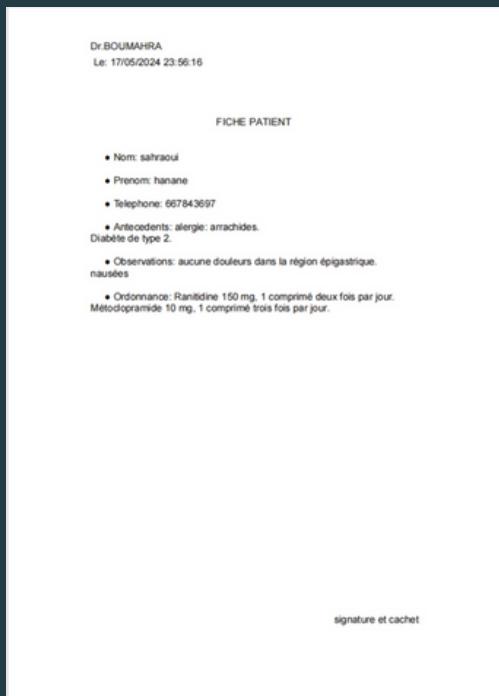
Patient pas trouvé!

**dans le cas contraire un message d'erreur sera affiché**

**dans le cas ou le patient a demandé une copie de sa fiche le médecin doit l'a imprimer on cliquons sur le bouton imprimer.la panne suivante sera affichée afin de traiter l'opération d'impression impression sur la fiche donner**



**avant d'imprimer la fiche elle doit être sauvegarder dans la forme choisi par le médecin ( pdf par exemple )**



après être sauvegardée ( avec l'apparition de la date et l'heure d'impression plus la zone de la signature et le cachet du médecin ) , la fiche patient sera imprimée , signée et cachet par le médecin et prêt à donner au patient qui l'a demandé .

# 08 Code java simple

Comme premier pas dans notre projet, nous avons commencé par écrire un simple code Java (sans interface graphique) qui contient chaque détail demandé dans l'énoncé du projet, tel que ce code qui représente un système de gestion médicale est à manipuler à partir du terminal. Ensuite, nous avons tenté de créer les interfaces graphiques comme une amélioration. Voici tout ce que nous avons dans notre code Java simple (qui sera inclus dans GitHub) :



## --->Interfaces :

### 1. **DossierMedical** :

- Définit les méthodes pour ajouter des consultations et des traitements, ainsi que pour obtenir les listes de consultations et de traitements.

### 2. **GestionRendezVous** :

- Définit les méthodes pour prendre et annuler des rendez-vous, ainsi que pour obtenir la liste des rendez-vous.

### 3. **GestionFichesPatients** :

- Définit les méthodes pour ajouter une fiche patient et une observation médicale, ainsi que pour obtenir la liste des fiches patients.

## --->Classes abstraites :

### 1. **Personne** :

- Classe abstraite représentant une personne avec un nom, un prénom et un téléphone.

## --->Classes concrètes :

### 1. **Consultation** :

- Représente une consultation médicale avec une **date** et un **résumé**.

### 2. **Traitemen**t :

- Représente un traitement médical avec un **médicament**, **une dose** et une **durée**.

### 3. **FichePatient** :

- Représente la fiche médicale d'un patient, incluant ses **antécédents** médicaux, ses **observations médicales**, ses **consultations** et ses **traitemen**ts.

- Implémente l'interface **DossierMedical**.

### 4. **ObservationMedicale** :

- Représente une observation médicale.

### 5. **Patient** :

- Représente un patient, héritant de la classe **Personne**.

### 6. **Medecin** :

- Représente un médecin, héritant de la classe Personne.
- Implémente les interfaces **GestionRendezVous** et **GestionFichesPatients**.

### 7. **Secretaire** :

- Représente une secrétaire, héritant de la classe **Personne**.
- Implémente les interfaces **GestionRendezVous** et **GestionFichesPatients**.

### 8. **RendezVous** :

- Représente un **rendez-vous** avec un patient à une date donnée.

## --->Classe principale Poo :

- Contient la méthode main qui implémente une interface utilisateur via la console.
- Permet de gérer les patients, leurs fiches, les observations médicales, les consultations, les traitements et les rendez-vous via des interactions utilisateur.

## ---Fonctionnalités---

### 1. Choix du rôle :

- L'utilisateur peut choisir entre les rôles de médecin ou de secrétaire, ou quitter le programme.

### 1. Actions du médecin :

- Ajouter un patient :
  - Entrer les informations du patient, créer une nouvelle fiche patient, ajouter une observation médicale, une consultation et un traitement.
- Voir les rendez-vous :
  - Afficher la liste des rendez-vous planifiés.
- Voir les fiches patients :
  - Afficher la fiche d'un patient en entrant son nom.

### 1. Actions du secrétaire :

- Voir les fiches patients :
  - Afficher la fiche d'un patient en entrant son nom.
- Gérer les rendez-vous :
  - Ajouter un rendez-vous pour un patient, annuler un rendez-vous ou voir la liste des rendez-vous planifiés.

## --->Fonctionnalités :

### 1. **Choix du rôle :**

- L'utilisateur peut choisir entre les rôles de médecin ou de secrétaire, ou quitter le programme.

### 2. **Actions du médecin :**

- Ajouter un patient :
  - Entrer les informations du patient, créer une nouvelle fiche patient, ajouter une observation médicale, une consultation et un traitement.
- Voir les rendez-vous :
  - Afficher la liste des rendez-vous planifiés.
- Voir les fiches patients :
  - Afficher la fiche d'un patient en entrant son nom.

### 3. **Actions du secrétaire :**

- Voir les fiches patients :
  - Afficher la fiche d'un patient en entrant son nom.
- Gérer les rendez-vous :
  - Ajouter un rendez-vous pour un patient, annuler un rendez-vous ou voir la liste des rendez-vous planifiés.

## --->Points clés :

### 1. **Gestion des dates :**

- Utilisation de **SimpleDateFormat** pour le formatage et l'analyse des dates.

### 2. **Interactivité :**

- Utilisation de Scanner pour la saisie des informations par l'utilisateur.

### 1. **Organisation des données :**

- Utilisation de **ArrayList** pour stocker les listes de fiches patients, de consultations, de traitements, d'observations médicales et de rendez-vous.

# 09 Conclusion

En conclusion, ce projet JAVA pour la gestion d'un cabinet médical offre une solution pratique et efficace pour simplifier les tâches administratives et améliorer la prise en charge des patients. En facilitant la prise de rendez-vous, la gestion des appels et le suivi des consultations, cette solution vise à optimiser l'organisation du cabinet médical. En outre, la gestion centralisée des informations facilite la coordination entre les différents membres du personnel, rendant les opérations du cabinet plus fluides et efficaces. L'amélioration de ces processus contribue à un environnement de travail plus harmonieux et à une meilleure satisfaction des patients.