

# Rapport de TP1

---

## Gestion des employés

---

*Réaliser par :*

Nouhaila GHANDOUR.

*En cadre par :*

Mme Layla ElKhrof

**Année université : 2024/2025**

Vous pouvez trouver le code complet sur mon dépôt GitHub :

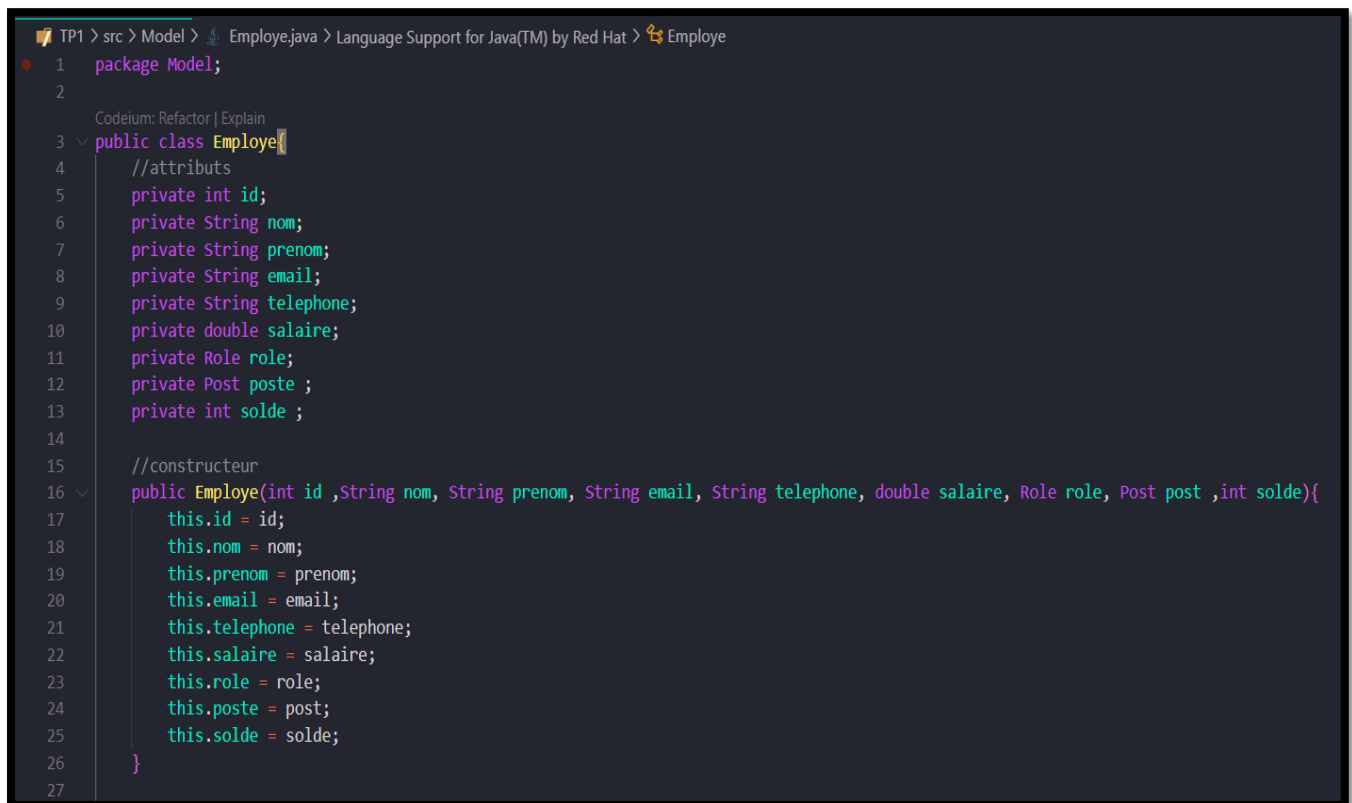
[NouhailaGhand/TP-java-](#)

## Codage:

### 1. Class Employé :

J'ai débuté par la création de la classe **Employé** ainsi que des énumérations **Poste** et **Rôle**. Ces dernières incluent les attributs, les getters, les setters et le constructeur.

L'ensemble a été regroupé dans un package nommer **Model**.



```
TP1 > src > Model > Employee.java > Language Support for Java(TM) by Red Hat > Employee
1 package Model;
2
3 Codeium: Refactor | Explain
4 public class Employe{
5     //attributs
6     private int id;
7     private String nom;
8     private String prenom;
9     private String email;
10    private String telephone;
11    private double salaire;
12    private Role role;
13    private Post poste ;
14    private int solde ;
15
16    //constructeur
17    public Employe(int id ,String nom, String prenom, String email, String telephone, double salaire, Role role, Post post ,int solde){
18        this.id = id;
19        this.nom = nom;
20        this.prenom = prenom;
21        this.email = email;
22        this.telephone = telephone;
23        this.salaire = salaire;
24        this.role = role;
25        this.poste = post;
26        this.solde = solde;
27    }
```

```

TP1 > src > Model > Employee.java > Language Support for Java
3 public class Employee{
27
28 //getters et setters
29
30 Codeium: Refactor | Explain | Generate Javadoc | X
31 public int getId(){
32     return id;
33 }
34
35 Codeium: Refactor | Explain | Generate Javadoc | X
36 public void setId(int id){
37     this.id = id;
38 }
39
40 Codeium: Refactor | Explain | Generate Javadoc | X
41 public String getNom() {
42     return nom;
43 }
44
45 Codeium: Refactor | Explain | Generate Javadoc | X
46 public void setNom(String nom) {
47     this.nom = nom;
48 }
49
50 Codeium: Refactor | Explain | Generate Javadoc | X
51 public String getPrenom() {
52     return prenom;
53 }
54
55 Codeium: Refactor | Explain | Generate Javadoc | X
56 public void setPrenom(String prenom) {
57     this.prenom = prenom;
58 }
59
60 Codeium: Refactor | Explain | Generate Javadoc | X
61 public String getEmail() {
62     return email;
63 }
64

```

```

TP1 > src > Model > Employee.java > Language Support for Java(TM) by
3 public class Employee{
20
30 Codeium: Refactor | Explain | Generate Javadoc | X
57 public void setEmail(String email) {
58     this.email = email;
59 }
60
61 Codeium: Refactor | Explain | Generate Javadoc | X
62 public String getTelephone() {
63     return telephone;
64 }
65
66 Codeium: Refactor | Explain | Generate Javadoc | X
67 public void setTelephone(String telephone) {
68     this.telephone = telephone;
69 }
70
71 Codeium: Refactor | Explain | Generate Javadoc | X
72 public double getSalaire() {
73     return salaire;
74 }
75
76 Codeium: Refactor | Explain | Generate Javadoc | X
77 public void setSalaire(double salaire) {
78     this.salaire = salaire;
79 }
80
81 Codeium: Refactor | Explain | Generate Javadoc | X
82 public Role getRole() {
83     return role;
84 }
85
86 Codeium: Refactor | Explain | Generate Javadoc | X
87 public void setRole(Role role) {
88     this.role = role;
89 }
90

```

```

TP1 > src > Model > Employee.java > Language Support for Java(
3 public class Employee{
84
85 Codeium: Refactor | Explain | Generate Javadoc | X
86 public Post getPost() {
87     return poste;
88 }
89
90 Codeium: Refactor | Explain | Generate Javadoc | X
91 public void setPost(Post post) {
92     this.poste = post;
93 }
94

```

```

1 package Model;
2
3 public enum Post {
4     PILOT, TEAM_LEADER, ENGINEER_STUDY_AND_DEVELOPMENT
5 }

```

```

1 package Model;
2
3 public enum Role {
4     ADMIN, EMPLOYEE, MANAGER
5 }

```

## 2. Class DBConnexion :

Ce code définit une classe **DBConnexion** dans le package **DAO** qui permet d'établir une connexion avec une base de données MySQL.

La classe utilise les informations de connexion telles que l'**URL** de la base de données, le **nom d'utilisateur** et le **mot de passe**.

La méthode **getConnexion ()** utilise l'**if condition** pour s'assurer qu'une seule instance de la connexion est créée. Si la connexion existe déjà, elle est retournée directement ; sinon, elle charge le driver JDBC, établit la connexion à la base de données via **DriverManager** et retourne cette connexion.

```

TP1 > src > DAO > DBConnexion.java
1 package DAO;
2
3 import java.sql.*;
4
5 class DBConnexion {
6     public static final String url = "jdbc:mysql://localhost:3306/tp";
7     public static final String user = "root";
8     public static final String password = "";
9     public static Connection conn = null;
10
11     public static Connection getConnexion() throws ClassNotFoundException {
12         if(conn != null){
13             return conn;
14         }
15         try {
16             Class.forName(className:"com.mysql.jdbc.Driver");
17             conn = DriverManager.getConnection(url, user, password);
18         } catch (SQLException e) {
19             throw new RuntimeException(message:"Error de connexion");
20         }
21
22         return conn;
23     }
24 }
25
26

```

### 3. Interface EmployeDAOI :

Ce code définit une **interface EmployeDAOI** dans le package **DAO**, qui sert de modèle pour les classes d'accès aux données (DAO).

```
app-gestion-java-main > src > DAO > EmployeeDAOI.java > Language
1 package DAO;
2
3 import Model.Employe;
4 import java.util.List;
5 public interface EmployeDAOI {
6     public boolean addEmploye(Employe e);
7     public void deleteEmploye(int id);
8     public void updateEmploye(Employe e);
9     public List<Employe> displayEmploye();
10 }
```

### 4. Class EmployeDAOImpl :

Ce code implémente la classe **EmployeDAOImpl**, qui fournit les fonctionnalités d'accès aux données pour la gestion des employés. Elle implémente l'interface **GenericDAOI<Employe>** et propose des méthodes pour **ajouter**, **supprimer**, **mettre à jour** et **afficher** des employés dans une base de données MySQL.

```
TP1 > src > DAO > EmployeeDAOImpl.java > Language Support for Java(TM) by Red Hat > EmployeeDAOImpl > add(Employe)
1
2 package DAO;
3 import Model.Employe;
4 import Model.Post;
5 import Model.Role;
6 import java.sql.PreparedStatement;
7 import java.sql.ResultSet;
8 import java.sql.SQLException;
9 import java.util.ArrayList;
10 import java.util.List;
11
12 Codeium: Refactor | Explain
13 public class EmployeDAOImpl implements GenericDAOI<Employe> {
14
15     // function of add Employe :
16     @Override
17     public void add(Employe e) {
18         String sql = "INSERT INTO employe (nom, prenom, email, telephone, salaire, role, poste, solde) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
19         try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
20             stmt.setString(parameterIndex:1, e.getNom());
21             stmt.setString(parameterIndex:2, e.getPrenom());
22             stmt.setString(parameterIndex:3, e.getEmail());
23             stmt.setString(parameterIndex:4, e.getTelephone());
24             stmt.setDouble(parameterIndex:5, e.getSalaire());
25             stmt.setString(parameterIndex:6, e.getRole().name());
26             stmt.setString(parameterIndex:7, e.getPost().name());
27             stmt.setInt(parameterIndex:8, e.getSolde());
28             stmt.executeUpdate();
29         } catch (SQLException exception) {
30             System.err.println("failed of add employe ");
31         } catch (ClassNotFoundException ex) {
32             System.err.println("failed connexion with data base");
33         }
34     }
35 }
```

TP1 > src > DAO > EmployeeDAOimpl.java > Language Support for Java(TM) by Red Hat > EmployeeDAOimpl > delete(int)

```
13 public class EmployeeDAOimpl implements GenericDAOI<Employee> {
35
36 // function of delete Employee :
   Codeium: Refactor | Explain | X
37 @Override
38 public void delete(int id) {
39     String sql = "DELETE FROM employee WHERE id = ?";
40     try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
41         stmt.setInt(parameterIndex:1,id);
42         stmt.executeUpdate();
43     } catch (SQLException exception) {
44         System.err.println(x:"failed of delete employee");
45     } catch (ClassNotFoundException ex) {
46         System.err.println(x:"failed connexion with data base");
47     }
48 }
49 }
```

```
50 // function of update Employee :
   Codeium: Refactor | Explain | X
51 @Override
52 public void update(Employee e) {
53     String sql = "UPDATE employee SET nom = ?, prenom = ?, email = ?, telephone = ?, salaire = ?, role = ?, poste = ? WHERE id = ?";
54     try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
55         stmt.setString(parameterIndex:1, e.getNom());
56         stmt.setString(parameterIndex:2, e.getPrenom());
57         stmt.setString(parameterIndex:3, e.getEmail());
58         stmt.setString(parameterIndex:4, e.getTelephone());
59         stmt.setDouble(parameterIndex:5, e.getSalaire());
60         stmt.setString(parameterIndex:6, e.getRole().name());
61         stmt.setString(parameterIndex:7, e.getPost().name());
62         stmt.setInt(parameterIndex:8,e.getId());
63         stmt.executeUpdate();
64     } catch (SQLException exception) {
65         System.err.println(x:"failed of update employee");
66     } catch (ClassNotFoundException ex) {
67         System.err.println(x:"failed connexion with data base");
68     }
69 }
70 }
```

```
71 // function of display Employee :
   Codeium: Refactor | Explain | X
72 @Override
73 public List<Employee> display() {
74     String sql = "SELECT * FROM employee";
75     List<Employee> Employes = new ArrayList<>();
76     try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
77         ResultSet re = stmt.executeQuery();
78         while (re.next()) {
79             int id = re.getInt(columnLabel:"id");
80             String nom = re.getString(columnLabel:"nom");
81             String prenom = re.getString(columnLabel:"prenom");
82             String email = re.getString(columnLabel:"email");
83             String telephone = re.getString(columnLabel:"telephone");
84             double salaire = re.getDouble(columnLabel:"salaire");
85             String role = re.getString(columnLabel:"role");
86             String poste = re.getString(columnLabel:"poste");
87             int solde = re.getInt(columnLabel:"solde");
88             Employee e = new Employee(id,nom, prenom, email, telephone, salaire, Role.valueOf(role), Post.valueOf(poste),solde);
89             Employes.add(e);
90         }
91         return Employes;
92     } catch (ClassNotFoundException ex) {
93         System.err.println(x:"failed connexion with data base");
94         return null;
95     } catch (SQLException ex) {
96         System.err.println(x:"failed of display employee");
97         return null;
98     }
99 }
100 }
```

## 5. Class EmployeModel :

Ce code définit la classe **EmployeModel**, qui sert de couche intermédiaire entre la **Vue** et le **DAO** pour la gestion des employés. Elle utilise une instance de **EmployeDAOimpl** pour accéder aux données et propose plusieurs méthodes métier :

- **addEmploye** : ajoute un employé après avoir vérifié des contraintes (salaire positif, format du téléphone et de l'email).
- **deleteEmploye** : supprime un employé via son ID.
- **updateEmploye** : met à jour les informations complètes d'un employé.
- **displayEmploye** : récupère et retourne la liste des employés.

```
TP1 > src > Model > EmployeeModel.java > Language Support for Java(TM) by Red Hat > EmployeeModel > addEmploye(int, String, String, String, String, double, Role, Post, int)
1 package Model;
2 import DAO.EmployeeDAOimpl;
3 import java.util.List;
4
5 Codeium: Refactor | Explain
6 public class EmployeModel {
7     private EmployeeDAOimpl dao;
8     public EmployeModel(EmployeeDAOimpl dao) {
9         this.dao = dao;
10    }
11
12    // fonction de add Employe :
13    Codeium: Refactor | Explain | X
14    public boolean addEmploye(int id ,String nom, String prenom, String email, String telephone, double salaire, Role role, Post post, int solde) {
15        if(salaire < 0 ){
16            System.out.println(x:"Erreur : le salaire doit etre positif.");
17            return false;
18        }
19
20        if(telephone.length() != 10){
21            System.out.println(x:"Erreur : le telephone doit etre 10 num.");
22            return false;
23        }
24        if(!email.contains(s:"@")){
25            System.out.println(x:"Erreur : le mail doit contenir le @.");
26            return false;
27        }
28
29        Employee e = new Employee(id,nom, prenom, email, telephone, salaire, role, post ,solde);
30
31        dao.add(e);
32
33        return true;
34    }
35}
```

```
34 // function of delete Employee :
35 Codeium: Refactor | Explain | X
36 public boolean deleteEmploye(int id){
37     dao.delete(id);
38     return true;
39 }
40
41 // function of update Employee :
42 Codeium: Refactor | Explain | X
43 public boolean updateEmploye(int id, String nom, String prenom, String email, String telephone, double salaire, Role role, Post post , int solde) {
44     Employee e = new Employee(id,nom, prenom, email, telephone, salaire, role, post,solde);
45     dao.update(e);
46     return true;
47 }
48
49 //function of display Employee :
50 Codeium: Refactor | Explain | X
51 public List<Employee> displayEmploye() {
52     List<Employee> Employees = dao.display();
53     return Employees;
54 }
```

## 6. Class ControllerEmploye :

Ce code définit la classe **EmployeController**, qui joue le rôle du **contrôleur** dans l'architecture **MVC** pour la gestion des employés. Elle coordonne les interactions entre la **Vue** (Employe\_View) et le **Modèle** (EmployeModel), tout en répondant aux actions de l'utilisateur.

```
9 public class EmployeController {
10
11     private final Employe_HolidayView View;
12     public static EmployeModel model_employe ;
13     public static int id = 0;
14     public static int oldselectedrow = -1;
15     public static boolean test = false;
16     String nom = "";
17     String prenom = "";
18     String email = "";
19     String telephone = "";
20     double salaire = 0;
21     Role role = null;
22     Post poste = null;
23     int solde = 0;
24     boolean updatereussi = false;
25
26     public EmployeController(Employe_HolidayView view, EmployeModel model) {
27         this.View = view;
28         this.model_employe = model;
29
30         // Action Listener :
31         View.getaddButton_employe().addActionListener(e -> addEmploye());
32         View.getdeleteButton_employe().addActionListener(e -> deleteEmploye());
33         View.getupdateButton_employe().addActionListener(e -> updateEmploye());
34         View.getdisplayButton_employe().addActionListener(e -> displayEmploye());
35         // Selection Listener :
36         Employe_HolidayView.Tableau.getSelectionModel().addListSelectionListener(e -> updateEmployebyselect());
37     }
38
39 }
```

### Fonctionnalités principales :

1. **Afficher les employés** : La méthode displayEmploye récupère la liste des employés via le modèle et met à jour la table dans la vue.

```
40 //function of display Employe :
41 Codeium: Refactor | Explain | X
42 public void displayEmploye() {
43     List<Employe> Employes = model_employe.displayEmploye();
44     if(Employes.isEmpty()){
45         View.afficherMessageErreur(message:"Aucun employe.");
46     }
47     DefaultTableModel tableModel = (DefaultTableModel) Employe_HolidayView.Tableau.getModel();
48     tableModel.setRowCount(0);
49     for(Employe e : Employes){
50         tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.getEmail(), e.getTelephone(), e.getSalaire(), e.getRole(), e.getPost(), e.getSolde()});
51     }
52     // remplir la liste des employes dans le holiday automatique :
53     View.remplir_les_employes();
54 }
```

2. **Ajouter un employé** : La méthode addEmploye ajoute un nouvel employé après avoir collecté les données du formulaire et les envoie au modèle pour traitement.



```

56 // function of add Employee :|
57 private void addEmployee() {
58     String nom = View.getNom();
59     String prenom = View.getPrenom();
60     String email = View.getEmail();
61     String telephone = View.getTelephone();
62     double salaire = View.getSalaire();
63     Role role = View.getRole();
64     Post poste = View.getPoste();
65
66     View.viderChamps_em();
67     boolean adresse = model_employe.addEmployee(id:0,nom, prenom, email, telephone, salaire, role, poste ,solde:25);
68
69     if(adresse == true){
70         View.afficherMessageSucces(message:"L'employe a bien ete ajoutee.");
71         displayEmploye();
72     }else{
73         View.afficherMessageErreur(message:"L'employe n'a pas ete ajoutee.");
74     }
75 }
76
77
78

```

3. **Supprimer un employé** : La méthode deleteEmploye supprime l'employé sélectionné dans la table.

```

79 // function of delete Employee :
80 private void deleteEmploye(){
81     int selectedrow = Employee_HolidayView.Tableau.getSelectedRow();
82     if(selectedrow == -1){
83         View.afficherMessageErreur(message:"Veuillez selectionner une ligne.");
84     }else{
85         int id = (int) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:0);
86         if(model_employe.deleteEmploye(id)){
87             View.afficherMessageSucces(message:"L'employe a bien ete supprimer.");
88             displayEmploye();
89         }else{
90             View.afficherMessageErreur(message:"L'employe n'a pas ete supprimer.");
91         }
92     }
93 }
94

```

4. **Mettre à jour un employé** : Le processus se divise en deux étapes :
- **updateEmployebyselect** : Récupère les informations de l'employé sélectionné et les affiche dans le formulaire.

```

96 // function one of fetch data employee by select and display it in the forme:
Codeium: Refactor | Explain | X
97 private void updateEmployeebyselect(){
98     int selectedrow = Employee_HolidayView.Tableau.getSelectedRow();
99
100     if (selectedrow == -1) {
101         return;
102     }
103     try{
104         id = (int) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:0);
105         nom = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:1);
106         prenom = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:2);
107         email = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:3);
108         telephone = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:4);
109         salaire = (double) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:5);
110         role = (Role) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:6);
111         poste = (Post) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:7);
112         solde = (int) Employee_HolidayView.Tableau.getValueAt(selectedrow, column:8);
113         View.remplaireChamps_em(id, nom, prenom, email, telephone, salaire, role, poste);
114         test = true;
115     }catch(Exception e){
116         View.afficherMessageErreur(message:"Erreur lors de la récupération des données");
117     }
118 }
119

```

- **updateEmployee** : Met à jour les informations après modification et validation.

```

120 // function two of update Employee by click update button :
Codeium: Refactor | Explain | X
121 private void updateEmployee(){
122     if (!test) {
123         View.afficherMessageErreur(message:"Veuillez d'abord sélectionner une ligne à modifier.");
124         return;
125     }
126     try {
127         nom = View.getNom();
128         prenom = View.getPrenom();
129         email = View.getEmail();
130         telephone = View.getTelephone();
131         salaire = View.getSalaire();
132         role = View.getRole();
133         poste = View.getPoste();
134
135         boolean updateSuccessful = model_employee.updateEmployee(id, nom, prenom, email, telephone, salaire, role, poste , solde);
136
137         if (updateSuccessful) {
138             test = false;
139             View.afficherMessageSucces(message:"L'employé a été modifié avec succès.");
140             displayEmployee();
141             View.viderChamps_em();
142         } else {
143             View.afficherMessageErreur(message:"Erreur lors de la mise à jour de l'employé.");
144         }
145     } catch (Exception e) {
146
147         View.afficherMessageErreur(message:"Erreur lors de la mise à jour");
148     }
149 }
150

```

## 7. Class EmployeView :

### Description des composants :

1. **JPanel général** (General) : Contient tous les autres panneaux et définit la mise en page.
2. **Table d'affichage** (Tableau) : Affiche les employés dans une table avec les colonnes : ID, Nom, Prénom, Email, Téléphone, Salaire, Rôle, Poste.
3. **Formulaire de saisie** (Forme) : Permet à l'utilisateur d'entrer les informations d'un employé, comme le nom, prénom, email, téléphone, salaire, rôle et poste.
  - Contient des JTextField pour les entrées de texte (nom, prénom, email, téléphone, salaire).
  - Contient des JComboBox pour sélectionner le rôle et le poste de l'employé.
4. **Boutons d'action** :
  - **Ajouter, Modifier, Supprimer, Afficher** : Ces boutons permettent à l'utilisateur de déclencher les actions correspondantes dans l'application.
5. **Méthodes** :
  - **getNom(), getPrenom(), getEmail(), etc.** : Renvoient les valeurs saisies dans les champs de texte.
  - **afficherMessageErreur(), afficherMessageSucces()** : Affichent des messages d'erreur ou de succès à l'utilisateur.
  - **viderChamps()** : Réinitialise tous les champs du formulaire après une action (ajout, mise à jour).
  - **remplaireChamps()** : Remplit le formulaire avec les données d'un employé lors de la sélection dans la table.
  - **testChampsVide()** : Vérifie si tous les champs obligatoires sont remplis.

```

app-gestion-java-main > src > View > EmployeeView.java > Language Support for Java(TM) by Red Hat > EmployeeView
1  package View;
2
3  import Model.Post;
4  import Model.Role;
5  import java.awt.*;
6  import javax.swing.*;
7  import javax.swing.table.DefaultTableModel;
8
9  public class EmployeeView extends JFrame {
10
11     private JPanel General = new JPanel();
12     private JPanel Display_Table = new JPanel();
13     private final JPanel Forme = new JPanel();
14     private JPanel panButton = new JPanel();
15
16     private JLabel label_nom = new JLabel(text:"Nom");
17     private JLabel label_prenom = new JLabel(text:"Prenom");
18     private JLabel label_email = new JLabel(text:"Email");
19     private JLabel label_tele = new JLabel(text:"Telephone");
20     private JLabel label_salaire = new JLabel(text:"Salaire");
21     private JLabel label_role = new JLabel(text:"Role");
22     private JLabel label_poste = new JLabel(text:"Poste");
23
24     private JTextField text_nom = new JTextField();
25     private JTextField text_prenom = new JTextField();
26     private JTextField text_email = new JTextField();
27     private JTextField text_tele = new JTextField();
28     private JTextField text_salaire = new JTextField();
29
30     private JComboBox<Role> roleComboBox = new JComboBox<>(Role.values());
31     private JComboBox<Post> posteComboBox = new JComboBox<>(Post.values());
32

```

```

app-gestion-java-main > src > View > EmployeeView.java > Language Support for Java(TM) by Red Hat > EmployeeView
9  public class EmployeeView extends JFrame {
32
33     private JButton addButton = new JButton(text:"Ajouter");
34     private JButton updateButton = new JButton(text:"Modifier");
35     private JButton deleteButton = new JButton(text:"Supprimer");
36     private JButton displayButton = new JButton(text:"Afficher");
37
38     JPanel pan0 = new JPanel(new BorderLayout());
39     public static String[] columnNames = {"ID", "Nom", "Prenom", "Email", "Téléphone", "Salaire", "Role", "Poste"};
40     public static DefaultTableModel tableModel = new DefaultTableModel(columnNames, rowCount:0);
41     public static JTable Tableau = new JTable(tableModel);
42
43     public EmployeeView() {
44         setTitle(title:"Gestion des employes");
45         setSize(width:1000, height:600);
46         setDefaultCloseOperation(EXIT_ON_CLOSE);
47         setLocationRelativeTo(c:null);
48
49         add(General);
50         General.setLayout(new BorderLayout());
51
52         General.add(Display_Table, BorderLayout.CENTER);
53
54         Tableau.setFillsViewportHeight(fillViewportHeight:true);
55         Dimension preferredSize = new Dimension(width:900, height:500);
56         Tableau.setPreferredScrollableViewportSize(preferredSize);
57         pan0.add(new JScrollPane(Tableau), BorderLayout.CENTER);
58         Display_Table.add(pan0);
59
60         General.add(panButton, BorderLayout.SOUTH);
61         panButton.add(addButton);
62         panButton.add(updateButton);
63         panButton.add(deleteButton);
64         panButton.add(displayButton);
65

```

```
app-gestion-java-main > src > View > EmployeeView.java > Language Support for Java(TM) by Red Hat > EmployeeView
9 public class EmployeeView extends JFrame {
43     public EmployeeView() {
66         General.add(Forme, BorderLayout.NORTH);
67         Forme.setLayout(new GridLayout(rows:7, cols:2, hgap:10, vgap:10));
68         Forme.add(label_nom);
69         Forme.add(text_nom);
70         Forme.add(label_prenom);
71         Forme.add(text_prenom);
72         Forme.add(label_email);
73         Forme.add(text_email);
74         Forme.add(label_tele);
75         Forme.add(text_tele);
76         Forme.add(label_salaire);
77         Forme.add(text_salaire);
78         Forme.add(label_role);
79         Forme.add(roleComboBox);
80         Forme.add(label_poste);
81         Forme.add(posteComboBox);
82
83         setVisible(b:true);
84     }
85
86     Codeium: Refactor | Explain | Generate Javadoc | X
87     public String getNom() {
88         return text_nom.getText();
89     }
90
91     Codeium: Refactor | Explain | Generate Javadoc | X
92     public JTable getTable() {
93         return (JTable) Display_Table.getComponent(n:0);
94     }
95
96     Codeium: Refactor | Explain | Generate Javadoc | X
97     public String getPrenom() {
98         return text_prenom.getText();
99     }
```

```
app-gestion-java-main > src > View > EmployeeView.java > Language Support for Java(TM) by Red Hat > EmployeeView
9 public class EmployeeView extends JFrame {
125
126     Codeium: Refactor | Explain | Generate Javadoc | X
127     public JButton getdeleteButton() {
128         return deleteButton;
129     }
130
131     Codeium: Refactor | Explain | Generate Javadoc | X
132     public JButton getdisplayButton() {
133         return displayButton;
134     }
135
136     Codeium: Refactor | Explain | Generate Javadoc | X
137     public void afficherMessageErreur(String message) {
138         JOptionPane.showMessageDialog(this, message, title:"Erreur", JOptionPane.ERROR_MESSAGE);
139     }
140
141     Codeium: Refactor | Explain | Generate Javadoc | X
142     public void afficherMessageSucces(String message) {
143         JOptionPane.showMessageDialog(this, message, title:"Succès", JOptionPane.INFORMATION_MESSAGE);
144     }
145 }
```

```
app-gestion-java-main > src > View > EmployeeView.java > Language Support for Java(TM) by Red Hat > EmployeeView
9 public class EmployeeView extends JFrame {
    Codeium: Refactor | Explain | Generate Javadoc | X
98 public String getEmail() {
99     return text_email.getText();
100 }
101
    Codeium: Refactor | Explain | Generate Javadoc | X
102 public String getTelephone() {
103     return text_tele.getText();
104 }
105
    Codeium: Refactor | Explain | Generate Javadoc | X
106 public double getSalaire() {
107     return Double.parseDouble(text_salaire.getText());
108 }
109
    Codeium: Refactor | Explain | Generate Javadoc | X
110 public Role getRole() {
111     return (Role) roleComboBox.getSelectedItem();
112 }
113
    Codeium: Refactor | Explain | Generate Javadoc | X
114 public Post getPoste() {
115     return (Post) posteComboBox.getSelectedItem();
116 }
117
    Codeium: Refactor | Explain | Generate Javadoc | X
118 public JButton getaddButton() {
119     return addButton;
120 }
121
    Codeium: Refactor | Explain | Generate Javadoc | X
122 public JButton getupdateButton() {
123     return updateButton;
124 }
125
```

```
Codeium: Refactor | Explain | Generate Javadoc | X
142 public void viderChamps() {
143     text_nom.setText("");
144     text_prenom.setText("");
145     text_email.setText("");
146     text_tele.setText("");
147     text_salaire.setText("");
148     roleComboBox.setSelectedIndex(anIndex:0);
149     posteComboBox.setSelectedIndex(anIndex:0);
150 }
151
    Codeium: Refactor | Explain | Generate Javadoc | X
152 public void remplireChamps(int id, String nom, String prenom, String email, String telephone, double salaire, Role role, Post poste) {
153     text_nom.setText(nom);
154     text_prenom.setText(prenom);
155     text_email.setText(email);
156     text_tele.setText(telephone);
157     text_salaire.setText(String.valueOf(salaire));
158     roleComboBox.setSelectedItem(role);
159     posteComboBox.setSelectedItem(poste);
160 }
161
```

```
162 public boolean testChampsVide() {
163     return text_nom.getText().equals(anObject:"") || text_prenom.getText().equals(anObject:"") || text_email.getText().equals(anObject:"") ||
164     text_tele.getText().equals(anObject:"") || text_salaire.getText().equals(anObject:"");
165 }
166
167
```

Gestion des employes

Nom

Prenom

Email

Telephone

Salaire

Role

ADMIN

Poste

PILOT

ID	Nom	Prenom	Email	Téléphone	Salaire	Role	Poste
28	datch	adam	adamdatch@gmail...	0645589347	10000.0	ADMIN	PILOT
29	koubi	mohamed-yassine	mohamedyassink...	0645593798	10000.0	ADMIN	PILOT

Ajouter

Modifier

Supprimer

Afficher

## Conclusion :

En conclusion, la classe **EmployeeView** implémente l'interface graphique pour la gestion des employés dans une application basée sur l'architecture **MVC**. Elle permet à l'utilisateur de visualiser, ajouter, modifier et supprimer des employés à travers une table et un formulaire intuitif. Les actions sont facilitées par des boutons et la validation des champs, garantissant une interaction fluide avec le système. Cette vue fonctionne en étroite collaboration avec le contrôleur pour gérer les données et afficher les résultats appropriés, offrant ainsi une interface claire et fonctionnelle pour la gestion des employés.