

**Université Cadi Ayyad
École Supérieure De Technologie-Safi
Département : Informatique
Filière : genie informatique first year**

Rapport du TP N°1 java avancée

Gestion des employés

Réalisé par : EN-NAHLI DOUAA

Encadré par : Mme. ELKHROF Leila

ANNÉE UNIVERSITAIRE : 2024/2025

Table des matières

Introduction	4
Outils & environnement de travail	5
1 Environnement de travail	5
2 Outils de travail	5
3 Language de Programmation	6
1 Réalisation	7
1 Création de la base de donnée	7
1.1 Script base de donnée	7
2 Architecture MVC (Model-View-Controller)	8
2.1 Model	8
2.2 DAO	11
2.3 Controller	13
2 Résultats	16
1 Tables Créées	16
2 Résultats de la partie View	16
3 Après Ajout	17
4 Après modification	18
5 Apres Suppression	19
3 Conclusion générale	21
4 Références	21

Table des figures

1	intellij idea logo	5
2	MySQL Workbench logo	5
3	xampp logo	5
4	java developpement kit logo	6
5	java logo	6
2.1	Tables de la base de donnée	16
2.2	Interface Utilisateur	17
2.3	Resultat Ajout	17
2.4	Affichage de l'Ajout	18
2.5	Resultat de modification	18
2.6	Affichage de modification	19
2.7	Resultat de suppression	19
2.8	affichage de tous ce qui est stocké à la base de donnée	20

Inroduction

Ce travail pratique (TP) se concentre sur le développement d'une application Java dédiée à la gestion des employés, en adoptant une structure basée sur l'architecture **MVC (Model-View-Controller)**. Ce projet s'inscrit dans le cadre de l'apprentissage des concepts fondamentaux de la programmation orientée objet (POO) et de la mise en œuvre d'interfaces graphiques avec la bibliothèque **Swing**. Il offre également une opportunité d'approfondir les compétences en conception logicielle et en organisation structurée du code pour garantir une séparation claire des responsabilités.

L'objectif principal est de développer une application intuitive et fonctionnelle permettant de manipuler des données d'employés. L'application est conçue pour gérer l'ajout, la modification, la suppression et l'affichage des informations des employés, tout en assurant une interface utilisateur fluide et interactive. Grâce à l'implémentation des principes de l'architecture MVC, ce projet garantit une maintenance simplifiée et une évolutivité de l'application.

Les fonctionnalités principales incluent :

- Ajout d'employés avec des informations complètes.
- Modification des données des employés.
- Suppression des employés.
- Affichage de la liste des employés.

Cette application vise non seulement à développer une solution technique pour la gestion des employés, mais aussi à démontrer la puissance des concepts de la programmation orientée objet combinée à une architecture bien définie. Elle constitue une étape fondamentale dans l'acquisition des compétences nécessaires pour des projets logiciels plus complexes à l'avenir.

Outils & environnement de travail

1 Environnement de travail



FIGURE 1 – intellij idea logo

- **IntelliJ idea** : est un environnement de développement intégré (IDE) développé par JetBrains, conçu principalement pour le développement en Java. Reconnu pour ses fonctionnalités intelligentes et sa grande efficacité, il prend également en charge de nombreux autres langages et frameworks comme Kotlin, Groovy, Scala, Python.

2 Outils de travail



FIGURE 2 – MySQL Workbench logo

- **MySQL Workbench** : un outil de travail graphique conçu pour faciliter la conception, l'administration, et la gestion des bases de données MySQL. Il fournit une interface utilisateur intuitive permettant de travailler avec des bases de données sans avoir à utiliser uniquement des commandes en ligne.



FIGURE 3 – xampp logo

- **xampp** : En parallèle, le projet vise à fournir des outils de gestion robustes pour le corps administratif, avec une fonctionnalité de multi-rôle, permettant à chaque agent d'accéder à un compte adapté à ses responsabilités spécifique



FIGURE 4 – java developpement kit logo

- **java developpement kit** : st un ensemble d'outils logiciels nécessaires pour développer des applications Java. Il inclut les composants essentiels pour coder, compiler, exécuter et déboguer des programmes Java.

3 Language de Programmation



FIGURE 5 – java logo

- **Java** : un langage de programmation orienté objet et une plateforme largement utilisée pour le développement d'applications logicielles. Il a été créé par Sun Microsystems (maintenant propriété d'Oracle) en 1995 et reste l'un des langages les plus populaires au monde, notamment pour les applications d'entreprise, le développement mobile (Android) et les applications web.

Réalisation

1 Création de la base de donnée

1.1 Script base de donnée

```
1 -- Cr ation de la base de donn es
2 CREATE DATABASE EmployeDB;
3
4 -- Utilisation de la base de donn es
5 USE EmployeDB;
6
7 -- Cr ation de la table des employ s
8 CREATE TABLE Employes (
9     id INT AUTO_INCREMENT PRIMARY KEY,
10    first_name VARCHAR(50),
11    last_name VARCHAR(50),
12    email VARCHAR(100),
13    phone_number VARCHAR(20),
14    salary DECIMAL(10, 2),
15    role VARCHAR(50),
16    poste VARCHAR(50)
17 );
18 -- Cr ation de la table Role
19 CREATE TABLE Role (
20     name varchar(30) NOT NULL
21 );
22 -- Cr ation de la table Poste
23 CREATE TABLE Poste (
24     name varchar(30) NOT NULL
25 );
```

Listing 1.1 – Script SQL de la base de données

- Ce script est écrit sur MySQL Workbench pour création la base de donnée pour être lié à au code via le driver JDBC pour garantir la gestion .

2 Architecture MVC (Model-View-Controller)

L'architecture MVC est un modèle de conception qui sépare les responsabilités au sein d'une application, facilitant ainsi la gestion et la maintenance du code. Elle repose sur trois composants principaux :

2.1 Model

Le modèle représente les données et la logique métier de l'application. Il gère l'accès aux données, effectue les calculs nécessaires et fournit les informations à la vue.

Employer

```
1 package model;
2
3 import enums.*;
4
5 public class Employer {
6
7     private int id;
8     private String firstName;
9     private String lastName;
10    private String email;
11    private int phoneNumber;
12    private double salary;
13    private Role role;
14    private Poste poste;
15
16    public Employer(int id, String firstName, String lastName, String email, int
phoneNumber, double salary, Role role, Poste poste) {
17        this.id = id;
18        this.firstName = firstName;
19        this.lastName = lastName;
20        this.email = email;
21        this.phoneNumber = phoneNumber;
22        this.salary = salary;
23        this.role = role;
24        this.poste = poste;
25    }
26
27    public int getId() {
28        return id;
29    }
30
31    public String getFirstName() {
32        return firstName;
33    }
34
35    public String getLastName() {
36        return lastName;
37    }
38
39    public String getEmail() {
40        return email;
```



```
41     }
42
43     public int getPhoneNumber() {
44         return phoneNumber;
45     }
46
47     public double getSalary() {
48         return salary;
49     }
50
51     public Role getRole() {
52         return role;
53     }
54
55     public Poste getPoste() {
56         return poste;
57     }
58 }
```

EmployerLogic

```
1 package model;
2 import enums.*;
3 import dao.EmployerDAO;
4 import java.util.List;
5 public class EmployerLogic {
6
7     private EmployerDAO dao;
8
9     public EmployerLogic (EmployerDAO dao) {
10         this.dao = dao;
11     }
12
13     public boolean addEmployer(int id, String firstName, String lastName, String email,
14 int phoneNumber, double salary, Role role, Poste poste) {
15
16         if ( isValidEmail(email) ) {
17             return dao.addEmployer( new Employer(
18                 id,
19                 firstName,
20                 lastName,
21                 email,
22                 phoneNumber,
23                 salary,
24                 role,
25                 poste
26             ));
27         }
28         return false;
29     }
30
31
32     public boolean updateEmployer(int id, String firstName, String lastName, String
33 email, int phoneNumber, double salary, Role role, Poste poste) {
```

```
34
35     if ( isValidEmail(email) ) {
36
37         Employer employer = new Employer(
38             id,
39             firstName,
40             lastName,
41             email,
42             phoneNumber,
43             salary,
44             role,
45             poste
46         );
47
48         return dao.updateEmployer(employer);
49     }
50
51     return false;
52 }
53
54 private boolean isValidEmail(String email) {
55     return email.contains("@gmail.com") ? true : false;
56 }
57
58
59 public boolean deleteEmployer(int id) {
60     return dao.deleteEmployer(id);
61 }
62
63 public List<Employer> getAllEmployers() {
64     return dao.getAllEmployers();
65 }
66
67 }
```

2.2 DAO

Le DAO est une couche qui permet de gérer l'interaction avec une base de données, en effectuant des opérations telles que la création, la lecture, la mise à jour et la suppression (CRUD) des données.

DBConnection

```
1 package dao;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class DBConnection {
8
9     private static final String URL = "jdbc:mysql://localhost:3306/
gestion_employes";
10    private static final String USERNAME = "root";
11    private static final String PASSWORD = "";
12
13    public static Connection getConnection() throws SQLException {
14        return DriverManager.getConnection(URL, USERNAME, PASSWORD);
15    }
16 }
```

EmployerDAO

```
1 package dao;
2
3 import java.sql.*;
4 import java.util.ArrayList;
5 import java.util.List;
6 import model.Employer;
7 import enums.Role;
8 import enums.Poste;
9
10 public class EmployerDAO implements EmployerInterface {
11
12     private Connection connection;
13
14     public EmployerDAO() {
15         try {
16             connection = DBConnection.getConnection();
17         } catch (SQLException connectionException) {
18             connectionException.printStackTrace();
19         }
20     }
21
22
23     @Override
24     public boolean addEmployer(Employer employer) {
25         try (PreparedStatement addStatement = connection.prepareStatement(
26             "INSERT INTO employers (first_name, last_name, email, phone, salary,
27             role, poste) VALUES (?, ?, ?, ?, ?, ?, ?)")) {
```

```
28         addStatement.setString(1, employer.getFirstName());
29         addStatement.setString(2, employer.getLastName());
30         addStatement.setString(3, employer.getEmail());
31         addStatement.setInt(4, employer.getPhoneNumber());
32         addStatement.setDouble(5, employer.getSalary());
33         addStatement.setString(6, employer.getRole().name());
34         addStatement.setString(7, employer.getPoste().name());
35
36         return addStatement.executeUpdate() > 0;
37
38     } catch (SQLException addException) {
39         addException.printStackTrace();
40         return false;
41     }
42 }
43
44
45 @Override
46 public boolean updateEmployer(Employer employer) {
47     try (PreparedStatement updateStatement = connection.prepareStatement("
UPDATE employers SET first_name = ?, last_name = ?, email = ?, phone = ?,
salary = ?, role = ?, poste = ? WHERE id = ?")) {
48
49         updateStatement.setString(1, employer.getFirstName());
50         updateStatement.setString(2, employer.getLastName());
51         updateStatement.setString(3, employer.getEmail());
52         updateStatement.setInt(4, employer.getPhoneNumber());
53         updateStatement.setDouble(5, employer.getSalary());
54         updateStatement.setString(6, employer.getRole().name());
55         updateStatement.setString(7, employer.getPoste().name());
56         updateStatement.setInt(8, employer.getId());
57
58         return updateStatement.executeUpdate() > 0;
59
60     } catch (SQLException updateException) {
61         updateException.printStackTrace();
62         return false;
63     }
64 }
65
66 @Override
67 public boolean deleteEmployer(int id) {
68     try (PreparedStatement deleteStatement = connection.prepareStatement("
DELETE FROM employers WHERE id = ?")) {
69
70         deleteStatement.setInt(1, id);
71         return deleteStatement.executeUpdate() > 0;
72
73     } catch (SQLException deleteException) {
74         return false;
75     }
76 }
77
78 @Override
79 public List<Employer> getAllEmployers() {
```

```

80     List<Employer> employers = new ArrayList<>();
81     try (ResultSet getResult = connection.prepareStatement("SELECT * FROM
employers").executeQuery()) {
82
83         while (getResult.next()) {
84             employers.add(new Employer(
85                 getResult.getInt("id"),
86                 getResult.getString("first_name"),
87                 getResult.getString("last_name"),
88                 getResult.getString("email"),
89                 getResult.getInt("phone"),
90                 getResult.getDouble("salary"),
91                 Role.valueOf(getResult.getString("role")),
92                 Poste.valueOf(getResult.getString("poste"))
93             ));
94         }
95
96     } catch (SQLException getException) {
97         getException.printStackTrace();
98     }
99     return employers;
100 }
101 }

```

EmployerInterface

```

1 package dao;
2
3 import java.util.List;
4 import model.Employer;
5
6 public interface EmployerInterface {
7     boolean addEmployer(Employer employer);
8     boolean updateEmployer(Employer employer);
9     boolean deleteEmployer(int id);
10    List<Employer> getAllEmployers();
11 }

```

2.3 Controller

Le contrôleur gère les actions de l'utilisateur. Il reçoit les événements de la vue, interagit avec le modèle pour effectuer des opérations (par exemple, ajout, modification, suppression de données), puis met à jour la vue en conséquence.

EmployerInterface

```

1 package controller;
2
3 import view.*;
4 import dao.*;
5 import model.*;
6 import enums.*;

```

```

7 import java.util.List;
8 import javax.swing.JOptionPane;
9 public class EmployerController {
10
11     private FormFrame frame;
12     private EmployerLogic employerLogic;
13
14     public EmployerController(FormFrame frame, EmployerLogic employerLogic) {
15         this.frame = frame;
16         this.employerLogic = employerLogic;
17
18         frame.getBtnPanel().getAddBtn().addActionListener(addEvent -> addEmployer
19         ());
20         frame.getBtnPanel().getUpdateBtn().addActionListener(updateEvent ->
21         updateEmployer());
22         frame.getBtnPanel().getRemoveBtn().addActionListener(deleteEvent ->
23         deleteEmployer());
24         loadEmployers();
25     }
26
27     private void addEmployer() {
28         try {
29             if (employerLogic.addEmployer(
30                 1,
31                 frame.getInPanel().getFirstNameField().getText(),
32                 frame.getInPanel().getLastNameField().getText(),
33                 frame.getInPanel().getEmailField().getText(),
34                 Integer.parseInt(frame.getInPanel().getTelephoneNumberField().
35                 getText()),
36                 Double.parseDouble(frame.getInPanel().getSalaryField().getText
37                 ()),
38                 Role.valueOf(frame.getInPanel().getSelectedRole().toString()),
39                 Poste.valueOf(frame.getInPanel().getSelectedPoste().toString())
40             )
41             )
42             {
43                 JOptionPane.showMessageDialog(frame, "Employer added successfully
44                 .");
45                 loadEmployers();
46             } else {
47                 JOptionPane.showMessageDialog(frame, "Failed to add employer.");
48             }
49         } catch (Exception e) {
50             JOptionPane.showMessageDialog(frame, "Invalid input: " + e.getMessage
51             ());
52         }
53     }
54
55     private void updateEmployer() {
56         try {
57             if (employerLogic.updateEmployer(
58                 frame.getListPanel().getSelectedRowId(),
59                 frame.getInPanel().getFirstNameField().getText(),
60                 frame.getInPanel().getLastNameField().getText(),

```

```
54         frame.getInPanel().getEmailField().getText(),
55         Integer.parseInt(frame.getInPanel().getTelephoneNumberField().
getText()),
56         Double.parseDouble(frame.getInPanel().getSalaryField().getText())
57     ,
58         Role.valueOf(frame.getInPanel().getSelectedRole().toString()),
59         Poste.valueOf(frame.getInPanel().getSelectedPoste().toString())
60     ))
61     {
62         JOptionPane.showMessageDialog(frame, "Employer updated successfully
.");
63         loadEmployers();
64     } else {
65         JOptionPane.showMessageDialog(frame, "Failed to update employer.");
66     }
67     } catch (Exception e) {
68         JOptionPane.showMessageDialog(frame, "Invalid input: " + e.getMessage());
69     }
70 }
71 private void deleteEmployer() {
72     try {
73         if (employerLogic.deleteEmployer(frame.getListPanel().getSelectedRowId())
) {
74             JOptionPane.showMessageDialog(frame, "Employer deleted successfully
.");
75             loadEmployers();
76         } else {
77             JOptionPane.showMessageDialog(frame, "Failed to delete employer.");
78         }
79     } catch (Exception e) {
80         JOptionPane.showMessageDialog(frame, "Invalid input: " + e.getMessage());
81     }
82 }
83
84 private void loadEmployers() {
85     frame.getListPanel().updateEmployerList(employerLogic.getAllEmployers());
86 }
87 }
```

Résultats

1 Tables Créées

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> employers	★	1	InnoDB	utf8mb4_general_ci	16,0 kio	-
<input type="checkbox"/> poste	★	3	InnoDB	utf8mb4_general_ci	16,0 kio	-
<input type="checkbox"/> role	★	2	InnoDB	utf8mb4_general_ci	16,0 kio	-
3 tables	Somme	6	InnoDB	utf8mb4_general_ci	48,0 kio	0 o

FIGURE 2.1 – Tables de la base de donnée

2 Résultats de la partie View

La couche View représente l'interface utilisateur de l'application et permet l'interaction entre l'utilisateur et le système. Dans ce projet, l'interface a été conçue avec le framework Swing en Java, qui fournit des composants graphiques riches et personnalisables.

The screenshot shows a window titled "Person Form" with the following fields and controls:

- First Name:
- Last Name:
- Email:
- Telephone Number:
- Salary:
- Role:
- Poste:

Below the form is a table with the following headers: Id, Nom, Prenom, Email, Salaire. The table body is empty.

At the bottom of the window are three buttons: Add, Remove, and Update.

FIGURE 2.2 – Interface Utilisateur

3 Après Ajout

Après l'ajout d'un employé, les informations saisies par l'utilisateur dans le panneau d'entrée sont validées et transmises à la couche Controller, qui communique avec la logique métier pour enregistrer les données. Une fois l'opération réussie, la liste des employés est automatiquement mise à jour dans le panneau d'affichage, reflétant les changements en temps réel.

The screenshot shows the "Person Form" window with the following data entered:

- First Name: douaa
- Last Name: ennahli
- Email: douaaenh@gmail.com
- Telephone Number: 0771567572
- Salary: 30000.0
- Role: EMPLOYER
- Poste: CHEF_DE_PROJET

A message box titled "Message" is displayed in the center, containing the text "Employer added successfully." and an "OK" button.

The table below the form is still empty.

FIGURE 2.3 – Resultat Ajout

Person Form

First Name: douaa

Last Name: ennahli

Email: douaaenh@gmail.com

Telephone Number: 0771567572

Salary: 30000.0

Role: EMPLOYER

Poste: CHEF_DE_PROJET

Id	Nom	Prenom	Email	Salaire
8	ennahli	douaa	douaaenh@gmail.com	30000.0

Add Remove Update

FIGURE 2.4 – Affichage de l’Ajout

4 Après modification

Après la mise à jour d’un employé, les nouvelles informations saisies par l’utilisateur dans le panneau d’entrée sont validées et transmises à la couche Controller, qui assure leur traitement via la logique métier.

Person Form

First Name: douaa

Last Name: ennahli

Email: douaaenh@gmail.com

Telephone Number: 0636037390

Salary: 60.000

Role: EMPLOYER

Poste: CHEF_DE_PROJET

Id	Nom	Prenom	Email	Salaire
8	ennahli	douaa	douaaenh@gmail.com	30000.0

Add Remove Update

Message: Employer updated successfully. OK

FIGURE 2.5 – Resultat de modification

Person Form

First Name: douaa

Last Name: ennahli

Email: douaaenh@gmail.com

Telephone Number: 0636037390

Salary: 60.000

Role: EMPLOYER

Poste: CHEF_DE_PROJET

Id	Nom	Prenom	Email	Salaire
8	ennahli	douaa	douaaenh@gmail.com	60.0

Add Remove Update

FIGURE 2.6 – Affichage de modification

5 Apres Suppression

Lorsqu'un employé est supprimé, l'utilisateur sélectionne l'employé concerné dans la liste affichée et confirme l'action en cliquant sur le bouton Supprimer. Cette demande est transmise à la couche Controller, qui s'assure de la suppression de l'enregistrement via la logique métier.

Person Form

First Name: hakima

Last Name: anouar

Email: hananeelhi@gmail.com

Telephone Number: 0619191515

Salary: 150.000

Role: EMPLOYER

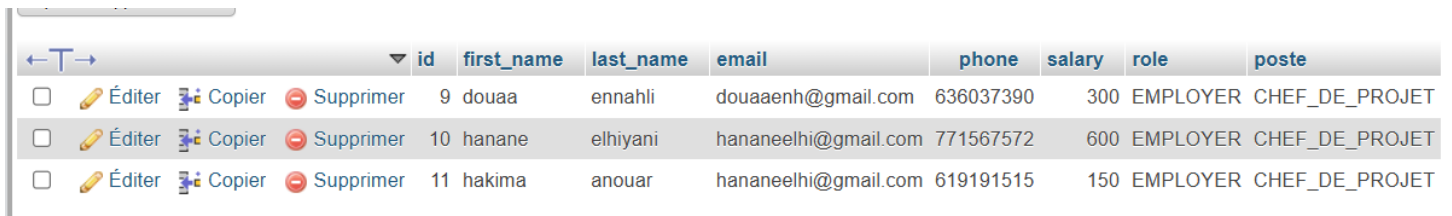
Poste: CHEF_DE_PROJET

Id	Nom	Prenom	Email	Salaire
9	ennahli	douaa	douaaenh@gmail.com	300.0
10	elhiyani	hanane	hananeelhi@gmail.com	600.0
11	anouar	hakima	hananeelhi@gmail.com	150.0

Add Remove Update

FIGURE 2.7 – Resultat de suppression

Tous ca est stocké dans la base de donnée voici le resultat.



				id	first_name	last_name	email	phone	salary	role	poste
<input type="checkbox"/>	Éditer	Copier	Supprimer	9	douaa	ennahli	douaaenh@gmail.com	636037390	300	EMPLOYER	CHEF_DE_PROJET
<input type="checkbox"/>	Éditer	Copier	Supprimer	10	hanane	elhiyani	hananeelhi@gmail.com	771567572	600	EMPLOYER	CHEF_DE_PROJET
<input type="checkbox"/>	Éditer	Copier	Supprimer	11	hakima	anouar	hananeelhi@gmail.com	619191515	150	EMPLOYER	CHEF_DE_PROJET

FIGURE 2.8 – affichage de tous ce qui est stocké à la base de donnée

Conclusion générale

En conclusion, ce Tp a permis de mettre en œuvre une application de gestion des employés en utilisant l'architecture ****MVC****. Grâce à cette structure, nous avons séparé clairement les responsabilités entre la logique métier, l'interface utilisateur et le traitement des données, garantissant ainsi une application modulaire, maintenable et extensible. L'intégration de fonctionnalités telles que l'ajout, la mise à jour et la suppression d'employés a renforcé notre compréhension des concepts de programmation orientée objet et de gestion d'interfaces graphiques en Java. Ce travail pratique illustre l'importance de structurer et d'organiser le code pour développer des applications robustes et performantes.

Références

java :

— <https://www.java.com/en/download/>

intellij idea :

— <https://www.jetbrains.com/idea/download/?ref=freeStuffDevsection=windows>

XAMPP :

— <https://www.apachefriends.org/fr/index.html>

jdk 23 :

— <https://www.oracle.com/java/technologies/downloads/>