Compte Rendu du TP 5:

JavaFX et accès aux base de données

Réalisé par : KHADIR Saad

Introduction:

Ce projet vise à concevoir et développer une application de gestion des étudiants et des groupes d'un établissement en s'appuyant sur Java et JavaFX pour l'interface utilisateur, ainsi que MySQL pour la gestion des données. L'application offre une interface conviviale permettant de gérer les informations des étudiants et des groupes, tout en intégrant des fonctionnalités essentielles telles que l'ajout, la modification, la suppression et la recherche de données.

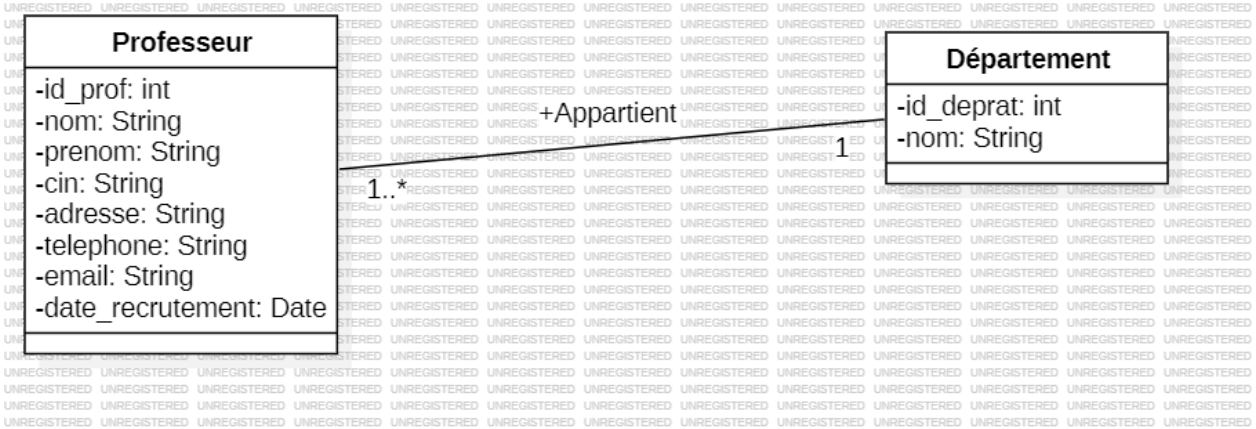
Objectifs :

L'objectif principal de ce TP est de concevoir et développer une application en Java utilisant une base de données relationnelle pour stocker et manipuler des données, tout en proposant une interface graphique conviviale grâce à JavaFX. Cette application permettra de gérer les étudiants et les groupes, avec des fonctionnalités telles que l'ajout, la recherche et la modification des données. Elle intégrera une connexion robuste à la base de données via JDBC en utilisant le design pattern Singleton, tout en mettant l'accent sur l'implémentation d'une interface graphique intuitive et fonctionnelle.

Diagramme de Classe:

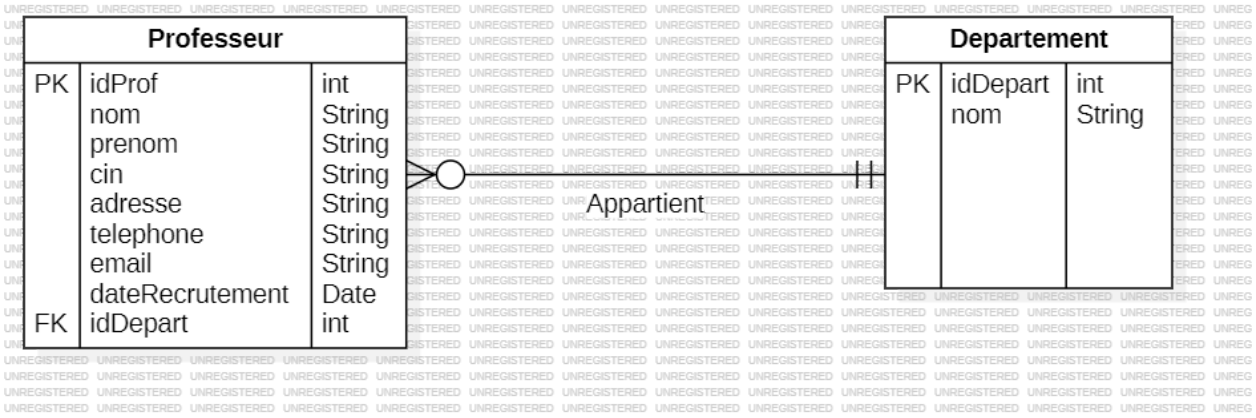
Un diagramme de classe a été élaboré pour modéliser les relations entre les entités **Professeur** et **Departement**. La classe **Professeur** représente les informations relatives à un professeur, tandis que la classe **Departement** regroupe plusieurs professeurs. Une relation d'association un-à-plusieurs a été définie entre **Departement** et **Professeur**, reflétant la structure où un département peut contenir plusieurs professeurs.

.



Modèle Logique des Données (MLD):

Le MLD traduit la structure des entités en tables relationnelles, où les relations entre **Professeur** et **Departement** sont modélisées à l'aide d'une clé étrangère, assurant ainsi l'intégrité référentielle entre les deux tables.



Classe 'Professeur':

```java
public class Professeur {
    private int idProf;
    private String nom;
    private String prenom;
    private String cin;
    private String adresse;
    private String telephone;
    private String email;
    private Date dateRecrutement;
    private int idDeprat;

    public Professeur() {
    }

    public Professeur(int idProf, String nom, String prenom, String cin, String adresse,
                      String telephone, String email, Date dateRecrutement, int idDeprat) {
        this.idProf = idProf;
        this.nom = nom;
        this.prenom = prenom;
        this.cin = cin;
        this.adresse = adresse;
        this.telephone = telephone;
        this.email = email;
        this.dateRecrutement = dateRecrutement;
        this.idDeprat = idDeprat;
    }
```

Classe 'Departement':

```java
public class Departement {
    private int idDepart;
    private String nomDepart;

    public Departement(int idDepart, String nomDepart) {
        this.idDepart = idDepart;
        this.nomDepart = nomDepart;
    }
    public Departement(){}

    public int getIdDepart() { return idDepart; }

    public void setIdDepart(int idDepart) { this.idDepart = idDepart; }

    public String getNomDepart() { return nomDepart; }

    public void setNomDepart(String nomDepart) { this.nomDepart = nomDepart; }

    @Override
    public String toString() {
        return "Departement{" +
                "idDepart=" + idDepart +
                ", nomDepart='" + nomDepart + '\'' +
                '}';
    }
}
```

Interface 'IMetier':

```java
public interface IMetier {
    void addProfesseur(Professeur professeur);
    boolean removeProfesseur(int id);
    Boolean updateProfesseur(Professeur professeur);
    List<Professeur> getAllProfesseurs();
    List<Professeur> searchProfesseurs(String keyword);

    void addDepartement(Departement departement);
    void removeDepartement(int id);
    void updateDepartement(Departement Departement);
    List<Departement> getAllDepartements();
    List<Professeur> getProfesseursByDepartement(int departementId);
}
```

Classe ‘SingletonConnexionDB’:

```java
public class SignletonConnexionDB {
    private static Connection connection;

    static {
        try {
            // Explicitly load the MySQL JDBC driver
            Class.forName( className: "com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to load MySQL JDBC driver", e);
        }
    }
    public SignletonConnexionDB() {
    }
    public static Connection getConnection() {
        if (connection == null) {
            try {
                connection = DriverManager.getConnection(
                        url: "jdbc:mysql://localhost:3306/gestionprofesseurs",
                        user: "root",
                        password: ""
                );
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return connection;
    }
}
```

Classe ‘MetierImpl’:

```java
public class MetierImpl implements IMetier{

    private SignletonConnexionDB signletonConnexionDB = new SignletonConnexionDB();
    private Connection connection = signletonConnexionDB.getConnection();

    @Override
    public void addProfesseur(Professeur professeur) {
        String query = "INSERT INTO Professeur(nom, prenom, cin, adresse, telephone," +
                " email, date_recrutement, id_deprat) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
        try (PreparedStatement ps = connection.prepareStatement(query)) {
            ps.setString( parameterIndex: 1, professeur.getNom());
            ps.setString( parameterIndex: 2, professeur.getPrenom());
            ps.setString( parameterIndex: 3, professeur.getCin());
            ps.setString( parameterIndex: 4, professeur.getAdresse());
            ps.setString( parameterIndex: 6, professeur.getEmail());
            ps.setString( parameterIndex: 5, professeur.getTelephone());
            ps.setDate( parameterIndex: 7,
                    new Date(professeur.getDateRecrutement().getTime()));
            ps.setInt( parameterIndex: 8, professeur.getIdDeprat());
            ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

```java
@Override
public boolean removeProfesseur(int id) {
    String query = "DELETE FROM Professeur WHERE id_prof = ?";
    try (PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setInt( parameterIndex: 1, id);
        ps.executeUpdate();
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}
```

```java
@Override
public Boolean updateProfesseur(Professeur professeur) {
    String query = "UPDATE Professeur SET nom = ?, prenom = ?, cin = ?," +
            " adresse = ?, telephone = ?, email = ?, date_recrutement = ?," +
            " id_deprat = ? WHERE id_prof = ?";
    try (PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setString( parameterIndex: 1, professeur.getNom());
        ps.setString( parameterIndex: 2, professeur.getPrenom());
        ps.setString( parameterIndex: 3, professeur.getCin());
        ps.setString( parameterIndex: 4, professeur.getAdresse());
        ps.setString( parameterIndex: 5, professeur.getTelephone());
        ps.setString( parameterIndex: 6, professeur.getEmail());
        ps.setDate( parameterIndex: 7,
                new Date(professeur.getDateRecrutement().getTime()));
        ps.setInt( parameterIndex: 8, professeur.getIdDeprat());
        ps.setInt( parameterIndex: 9, professeur.getIdProf());
        ps.executeUpdate();
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}
```

```java
public Professeur getProfesseurById(String id){
    Professeur professeur = null; // Initialize to null in case no result is fo
    int idProf = Integer.parseInt(id); // Convert string id to integer
    String query = "SELECT * FROM Professeur WHERE id_prof = ?"; // Use paramet

    try (PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setInt( parameterIndex: 1, idProf); // Set the id value for the query
        System.out.println("hey");
        try (ResultSet rs = ps.executeQuery()) { // Execute the query
            if (rs.next()) { // If there's at least one result
                professeur = new Professeur(
                        rs.getInt( columnLabel: "id_prof"),
                        rs.getString( columnLabel: "nom"),
                        rs.getString( columnLabel: "prenom"),
                        rs.getString( columnLabel: "cin"),
                        rs.getString( columnLabel: "adresse"),
                        rs.getString( columnLabel: "telephone"),
                        rs.getString( columnLabel: "email"),
                        rs.getDate( columnLabel: "date_recrutement"),
                        rs.getInt( columnLabel: "id_deprat")
                );
            }
        }
    } catch (SQLException e) {
        e.printStackTrace(); // Log the exception if any SQL error occurs
    }
```

```java
@Override
public List<Professeur> getAllProfesseurs() {
    List<Professeur> professeurs = new ArrayList<>();
    String query = "SELECT * FROM Professeur";
    try (Statement st = connection.createStatement();
         ResultSet rs = st.executeQuery(query)) {
        while (rs.next()) {
            professeurs.add(new Professeur(
                    rs.getInt( columnLabel: "id_prof"),
                    rs.getString( columnLabel: "nom"),
                    rs.getString( columnLabel: "prenom"),
                    rs.getString( columnLabel: "cin"),
                    rs.getString( columnLabel: "adresse"),
                    rs.getString( columnLabel: "telephone"),
                    rs.getString( columnLabel: "email"),
                    rs.getDate( columnLabel: "date_recrutement"),
                    rs.getInt( columnLabel: "id_deprat")
            ));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return professeurs;
}
```

```java
@Override
public List<Professeur> searchProfesseurs(String keyword) {
    List<Professeur> professeurs = new ArrayList<>();
    String query = "SELECT * FROM Professeur WHERE nom LIKE ? OR prenom LIKE ?";
    try (PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setString( parameterIndex: 1,  x: "%" + keyword + "%");
        ps.setString( parameterIndex: 2,  x: "%" + keyword + "%");
        try (ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                professeurs.add(new Professeur(
                        rs.getInt( columnLabel: "id_prof"),
                        rs.getString( columnLabel: "nom"),
                        rs.getString( columnLabel: "prenom"),
                        rs.getString( columnLabel: "cin"),
                        rs.getString( columnLabel: "adresse"),
                        rs.getString( columnLabel: "telephone"),
                        rs.getString( columnLabel: "email"),
                        rs.getDate( columnLabel: "date_recrutement"),
                        rs.getInt( columnLabel: "id_deprat")
                ));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return professeurs;
}
```

```java
@Override
public void addDepartement(Departement departement) {
    String query = "INSERT INTO Departement(nom) VALUES (?)";
    try (PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setString( parameterIndex: 1, departement.getNomDepart());
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Override
public void removeDepartement(int id) {
    String query = "DELETE FROM Departement WHERE id_deprat = ?";
    try (PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setInt( parameterIndex: 1, id);
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```java
@Override
public void updateDepartement(Departement departement) {
    String query = "UPDATE Departement SET nom = ? WHERE id_deprat = ?";
    try (PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setString( parameterIndex: 1, departement.getNomDepart());
        ps.setInt( parameterIndex: 2, departement.getIdDepart());
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Override
public List<Departement> getAllDepartements() {
    List<Departement> departements = new ArrayList<>();
    String query = "SELECT * FROM Departement";
    try (Statement st = connection.createStatement();
         ResultSet rs = st.executeQuery(query)) {
        while (rs.next()) {
            departements.add(new Departement(
                    rs.getInt( columnLabel: "id_deprat"),
                    rs.getString( columnLabel: "nom")
            ));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return departements;
```

```java
@Override
public List<Professeur> getProfesseursByDepartement(int departementId) {
    List<Professeur> professeurs = new ArrayList<>();
    String query = "SELECT * FROM Professeur WHERE id_deprat = ?";
    try (PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setInt( parameterIndex: 1, departementId);
        try (ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                professeurs.add(new Professeur(
                        rs.getInt( columnLabel: "id_prof"),
                        rs.getString( columnLabel: "nom"),
                        rs.getString( columnLabel: "prenom"),
                        rs.getString( columnLabel: "cin"),
                        rs.getString( columnLabel: "adresse"),
                        rs.getString( columnLabel: "telephone"),
                        rs.getString( columnLabel: "email"),
                        rs.getDate( columnLabel: "date_recrutement"),
                        rs.getInt( columnLabel: "id_deprat")
                ));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return professeurs;
}
```

La classe MetierImpl implémente l'interface IMetier et contient les fonctionnalités principales pour gérer les entités Professeur et Departement dans une application utilisant une base de données relationnelle. Elle repose sur une connexion JDBC gérée par le SingletonConnexionDB pour garantir une instance unique de la connexion à la base de données.

Les méthodes fournies permettent d'effectuer des opérations CRUD (Create, Read, Update, Delete) sur les entités. Pour la gestion des professeurs, la classe propose des fonctionnalités pour ajouter, modifier, supprimer, récupérer un professeur par son ID, rechercher des professeurs

via un mot-clé, et lister tous les professeurs ou ceux appartenant à un département spécifique. Ces opérations utilisent des requêtes SQL, avec des PreparedStatement pour prévenir les risques liés aux injections SQL.

De manière similaire, pour la gestion des départements, les méthodes permettent d'ajouter, modifier, supprimer et lister tous les départements. Une méthode supplémentaire permet d'obtenir la liste des professeurs associés à un département spécifique.

L'implémentation met l'accent sur la robustesse et la sécurité grâce à l'utilisation des bonnes pratiques, comme les blocs *try-with-resources* pour gérer automatiquement les ressources (connexion, statements, résultats). Enfin, les entités manipulées, telles que Professeur et Departement, sont entièrement modélisées, permettant un traitement fluide des données entre l'application et la base de données.

Interface Graphique avec JavaFX:

Une interface JavaFX a été développée pour offrir une gestion conviviale des interactions avec l'utilisateur. Cette interface comprend un formulaire permettant d'ajouter, de modifier ou de supprimer un professeur, ainsi qu'un autre formulaire dédié à l'ajout de départements. Les données, comme la liste des professeurs et des départements, sont affichées de manière dynamique dans une **TableView**.

Afin de gérer efficacement les actions de l'utilisateur, telles que les clics sur les boutons ou les saisies dans les champs de texte, un contrôleur JavaFX a été mis en place. Ce contrôleur sert d'intermédiaire entre l'interface graphique et le backend, en appelant les méthodes nécessaires pour exécuter la logique métier. Par exemple, il permet de récupérer une liste de professeurs depuis la base de données, de rechercher un professeur précis, ou encore d'ajouter un nouveau professeur tout en mettant à jour l'interface en conséquence.

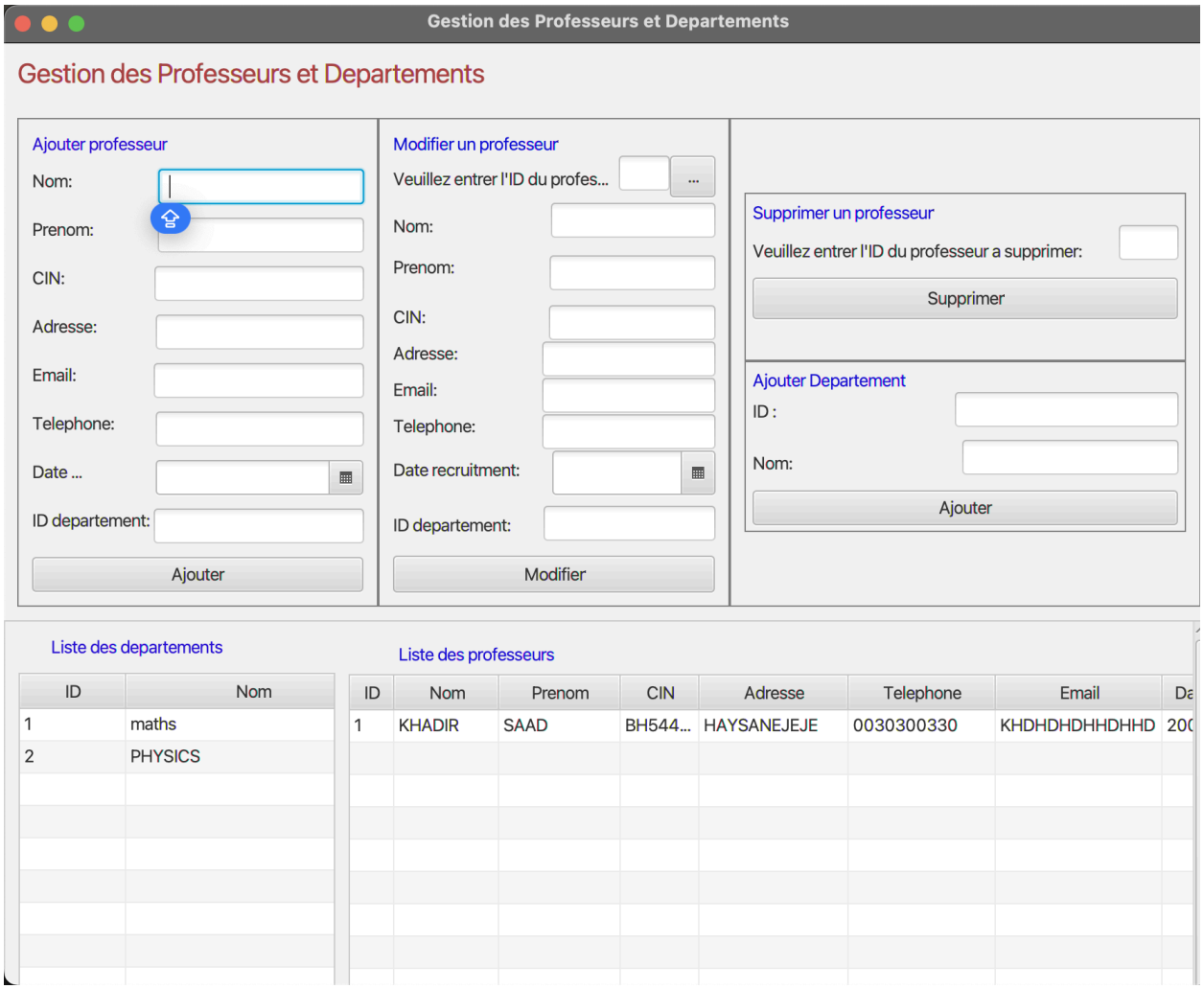
```java
@FXML
public void initialize() {
    setupTables();

    addProfessorButton.setOnAction(this::handleAddProfessor);
    searchButton.setOnAction(this::handleSearchButtonAction);
    modifyButton.setOnAction(this::handleModifyButtonAction);
    ajouterDepartementButton.setOnAction(this::handleAjouterDepartement);
    supprimerProfButton.setOnAction(this::handleDeleteProfessor);
}
```

```java
private void handleSearchButtonAction(ActionEvent event) {  1 usage
    String id = idProfField.getText(); // Get the ID entered by the user

    if (id == null || id.isEmpty()) {
        showAlert(Alert.AlertType.ERROR, title: "Error", message: "Please enter a valid Professor ID.");
        return;
    }

    // Retrieve the professor's information
    Professeur professor = metier.getProfesseurById(id);

    if (professor == null) {
        showAlert(Alert.Al FR ype.ERROR, title: "Error", message: "Professor not found with the given ID.");
        return;
    }

    // Populate the form fields wit
    nomField1.setText(professor.get
    prenomField1.setText(professor.
    cinField1.setText(professor.getCin());
    adresseField1.setText(professor.getAdresse());
    emailField1.setText(professor.getEmail());
    telephoneField1.setText(professor.getTelephone());
    System.out.println(LocalDate.parse(professor.getDateRecrutement().toString()));
    dateRecruitmentPicker1.setValue(LocalDate.parse(professor.getDateRecrutement().toString()));
    idDepartementField1.setText(String.valueOf(professor.getIdDeprat()));
```

```
javafx.scene.control.Alert.AlertType
public static final Alert.AlertType ERROR
// Enum constant ordinal: 4
Maven: org.openjfx:javafx-contro...h64.jar)
```

```java
    telephoneField1.setText(professor.getTelephone());
    System.out.println(LocalDate.parse(professor.getDateRecrutement().toString()));
    dateRecruitmentPicker1.setValue(LocalDate.parse(professor.getDateRecrutement().toString()));
    idDepartementField1.setText(String.valueOf(professor.getIdDeprat()));

    // Enable the "Modifier" button
    modifyButton.setDisable(false);
}
```

- **Interface de gestion de professeurs et departements:**



Conclusion:

Ce projet (TP) nous a permis de développer une application Java complète pour la gestion des professeurs et des départements en adoptant une approche modulaire et orientée objet. Les étapes de conception, de développement et de test ont été organisées de manière méthodique afin d'assurer une application fonctionnelle et extensible.