Một chương trình Java MVC CRUD với JFrame, sử dụng Maven trên IntelliJ, kết nối MySQL database và chuyển Department từ JTextField sang JComboBox. Chương trình sẽ bao gồm các tác vụ Create, Read, Update, Delete (CRUD) cho bảng Employee với quan hệ 1-N tới bảng Department. Dưới đây là hướng dẫn chi tiết và mã nguồn.

**1. Thiết lập dự án Maven trên IntelliJ**

**Bước 1: Tạo dự án Maven**

1. Mở IntelliJ IDEA.
2. Chọn File > New > Project.
3. Chọn Maven > Nhấn Next.
4. Đặt GroupId (ví dụ: com.example), ArtifactId (ví dụ: EmployeeManagement), nhấn Next và Finish.

**Bước 2: Cấu hình pom.xml**

Thêm các dependency cần thiết cho MySQL Connector và Swing (nếu cần). Dưới đây là file pom.xml:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>EmployeeManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

<dependencies>

*<!-- MySQL Connector -->*

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.33</version>

</dependency>

</dependencies>

</project>

**2. Thiết kế cơ sở dữ liệu (MySQL)**

Dựa trên yêu cầu, giả định đã có cơ sở dữ liệu EMS với hai bảng Department và Employee. Dưới đây là script SQL để tạo bảng:

CREATE DATABASE **EMS**;

USE **EMS**;

CREATE TABLE **Department** (

deptId INT PRIMARY KEY AUTO\_INCREMENT,

deptName VARCHAR(100) NOT NULL

);

CREATE TABLE **Employee** (

empId INT PRIMARY KEY AUTO\_INCREMENT,

fullName VARCHAR(100) NOT NULL,

gender VARCHAR(10) NOT NULL,

birthDate DATE NOT NULL,

deptId INT,

salary DOUBLE NOT NULL,

**FOREIGN** **KEY** (deptId) **REFERENCES** **Department**(deptId)

);

*-- Thêm dữ liệu mẫu*

INSERT INTO **Department** (deptName) VALUES ('Accounting De.'), ('IT');

INSERT INTO **Employee** (fullName, gender, birthDate, deptId, salary) VALUES

('Jane Doe', 'Female', '1989-01-01', 1, 2.0E7),

('Dao Nguyen', 'Male', '1967-05-04', 2, 1000000.0),

('Thang Pham', 'Male', '1999-01-01', 1, 2.5E7),

('Duy Le Lap', 'Male', '1995-12-12', 2, 2.5E7);

**3. Cấu trúc thư mục dự án**

Tạo cấu trúc thư mục theo mô hình MVC:

src/main/java/com/example/

├── model/

│ ├── Employee.java

│ ├── Department.java

├── view/

│ ├── EmployeeView.java

├── controller/

│ ├── EmployeeController.java

├── dao/

│ ├── EmployeeDAO.java

│ ├── DepartmentDAO.java

├── util/

│ ├── DatabaseConnection.java

└── Main.java

**4. Viết mã nguồn**

**4.1. Database Connection (DatabaseConnection.java)**

Tạo class để kết nối tới MySQL database.

package com.example.util;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class **DatabaseConnection** {

private static final String URL = "jdbc:mysql://localhost:3306/EMS";

private static final String USER = "root"; *// Thay bằng username MySQL của bạn*

private static final String PASSWORD = "password"; *// Thay bằng password MySQL của bạn*

public static Connection getConnection() throws SQLException {

return DriverManager.getConnection(URL, USER, PASSWORD);

}

}

**4.2. Model (Employee.java và Department.java)**

**Department.java**:

package com.example.model;

public class **Department** {

private int deptId;

private String deptName;

public Department(int deptId, String deptName) {

this.deptId = deptId;

this.deptName = deptName;

}

public int getDeptId() {

return deptId;

}

public String getDeptName() {

return deptName;

}

@Override

public String toString() {

return deptName;

}

}

**Employee.java**:

package com.example.model;

import java.util.Date;

public class **Employee** {

private int empId;

private String fullName;

private String gender;

private Date birthDate;

private Department department;

private double salary;

public Employee(int empId, String fullName, String gender, Date birthDate, Department department, double salary) {

this.empId = empId;

this.fullName = fullName;

this.gender = gender;

this.birthDate = birthDate;

this.department = department;

this.salary = salary;

}

public int getEmpId() {

return empId;

}

public String getFullName() {

return fullName;

}

public String getGender() {

return gender;

}

public Date getBirthDate() {

return birthDate;

}

public Department getDepartment() {

return department;

}

public double getSalary() {

return salary;

}

}

**4.3. DAO (DepartmentDAO.java và EmployeeDAO.java)**

**DepartmentDAO.java**:

package com.example.dao;

import com.example.model.Department;

import com.example.util.DatabaseConnection;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class **DepartmentDAO** {

public List<Department> getAllDepartments() throws SQLException {

List<Department> departments = new ArrayList<>();

String query = "SELECT \* FROM Department";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

departments.add(new Department(rs.getInt("deptId"), rs.getString("deptName")));

}

}

return departments;

}

public Department getDepartmentById(int deptId) throws SQLException {

String query = "SELECT \* FROM Department WHERE deptId = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, deptId);

try (ResultSet rs = stmt.executeQuery()) {

if (rs.next()) {

return new Department(rs.getInt("deptId"), rs.getString("deptName"));

}

}

}

return null;

}

}

**EmployeeDAO.java**:

package com.example.dao;

import com.example.model.Department;

import com.example.model.Employee;

import com.example.util.DatabaseConnection;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class **EmployeeDAO** {

private final DepartmentDAO departmentDAO = new DepartmentDAO();

public void addEmployee(Employee employee) throws SQLException {

String query = "INSERT INTO Employee (fullName, gender, birthDate, deptId, salary) VALUES (?, ?, ?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, employee.getFullName());

stmt.setString(2, employee.getGender());

stmt.setDate(3, new java.sql.Date(employee.getBirthDate().getTime()));

stmt.setInt(4, employee.getDepartment().getDeptId());

stmt.setDouble(5, employee.getSalary());

stmt.executeUpdate();

}

}

public void updateEmployee(Employee employee) throws SQLException {

String query = "UPDATE Employee SET fullName = ?, gender = ?, birthDate = ?, deptId = ?, salary = ? WHERE empId = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, employee.getFullName());

stmt.setString(2, employee.getGender());

stmt.setDate(3, new java.sql.Date(employee.getBirthDate().getTime()));

stmt.setInt(4, employee.getDepartment().getDeptId());

stmt.setDouble(5, employee.getSalary());

stmt.setInt(6, employee.getEmpId());

stmt.executeUpdate();

}

}

public void deleteEmployee(int empId) throws SQLException {

String query = "DELETE FROM Employee WHERE empId = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, empId);

stmt.executeUpdate();

}

}

public List<Employee> getAllEmployees() throws SQLException {

List<Employee> employees = new ArrayList<>();

String query = "SELECT \* FROM Employee";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Department dept = departmentDAO.getDepartmentById(rs.getInt("deptId"));

Employee emp = new Employee(

rs.getInt("empId"),

rs.getString("fullName"),

rs.getString("gender"),

rs.getDate("birthDate"),

dept,

rs.getDouble("salary")

);

employees.add(emp);

}

}

return employees;

}

public List<Employee> searchByDepartment(int deptId) throws SQLException {

List<Employee> employees = new ArrayList<>();

String query = "SELECT \* FROM Employee WHERE deptId = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, deptId);

try (ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Department dept = departmentDAO.getDepartmentById(rs.getInt("deptId"));

Employee emp = new Employee(

rs.getInt("empId"),

rs.getString("fullName"),

rs.getString("gender"),

rs.getDate("birthDate"),

dept,

rs.getDouble("salary")

);

employees.add(emp);

}

}

}

return employees;

}

public List<Employee> searchBySalary(double salary) throws SQLException {

List<Employee> employees = new ArrayList<>();

String query = "SELECT \* FROM Employee WHERE salary = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setDouble(1, salary);

try (ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Department dept = departmentDAO.getDepartmentById(rs.getInt("deptId"));

Employee emp = new Employee(

rs.getInt("empId"),

rs.getString("fullName"),

rs.getString("gender"),

rs.getDate("birthDate"),

dept,

rs.getDouble("salary")

);

employees.add(emp);

}

}

}

return employees;

}

}

**4.4. View (EmployeeView.java)**

Tạo giao diện JFrame giống như hình, nhưng thay JTextField của Department thành JComboBox.

package com.example.view;

import com.example.model.Department;

import com.example.model.Employee;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.text.SimpleDateFormat;

import java.util.List;

public class **EmployeeView** extends **JFrame** {

private JTextField txtEmpId, txtFullName, txtGender, txtBirthDate, txtSalary;

private JComboBox<Department> cbDepartment;

private JButton btnAdd, btnUpdate, btnDelete, btnLoad;

private JTable table;

private DefaultTableModel tableModel;

public EmployeeView() {

setTitle("EMPLOYEE MANAGEMENT");

setSize(800, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

*// Panel nhập liệu*

JPanel inputPanel = new JPanel(new GridLayout(6, 2));

inputPanel.add(new JLabel("EMPLOYEE ID:"));

txtEmpId = new JTextField();

txtEmpId.setEditable(false);

inputPanel.add(txtEmpId);

inputPanel.add(new JLabel("FULL NAME:"));

txtFullName = new JTextField();

inputPanel.add(txtFullName);

inputPanel.add(new JLabel("GENDER:"));

txtGender = new JTextField();

inputPanel.add(txtGender);

inputPanel.add(new JLabel("BIRTH DATE:"));

txtBirthDate = new JTextField();

inputPanel.add(txtBirthDate);

inputPanel.add(new JLabel("DEPARTMENT:"));

cbDepartment = new JComboBox<>();

inputPanel.add(cbDepartment);

inputPanel.add(new JLabel("SALARY:"));

txtSalary = new JTextField();

inputPanel.add(txtSalary);

*// Panel nút bấm*

JPanel buttonPanel = new JPanel(new GridLayout(1, 3));

btnAdd = new JButton("ADD");

btnUpdate = new JButton("UPDATE");

btnDelete = new JButton("DELETE");

btnLoad = new JButton("LOAD");

buttonPanel.add(btnAdd);

buttonPanel.add(btnUpdate);

buttonPanel.add(btnDelete);

buttonPanel.add(btnLoad);

*// Bảng hiển thị dữ liệu*

String[] columns = {"ID", "FULL NAME", "GENDER", "BIRTH DATE", "DEPARTMENT", "SALARY"};

tableModel = new DefaultTableModel(columns, 0);

table = new JTable(tableModel);

JScrollPane scrollPane = new JScrollPane(table);

*// Thêm các thành phần vào JFrame*

add(inputPanel, BorderLayout.NORTH);

add(buttonPanel, BorderLayout.CENTER);

add(scrollPane, BorderLayout.SOUTH);

}

*// Getters cho Controller*

public JTextField getTxtEmpId() {

return txtEmpId;

}

public JTextField getTxtFullName() {

return txtFullName;

}

public JTextField getTxtGender() {

return txtGender;

}

public JTextField getTxtBirthDate() {

return txtBirthDate;

}

public JComboBox<Department> getCbDepartment() {

return cbDepartment;

}

public JTextField getTxtSalary() {

return txtSalary;

}

public JButton getBtnAdd() {

return btnAdd;

}

public JButton getBtnUpdate() {

return btnUpdate;

}

public JButton getBtnDelete() {

return btnDelete;

}

public JButton getBtnLoad() {

return btnLoad;

}

public JTable getTable() {

return table;

}

public DefaultTableModel getTableModel() {

return tableModel;

}

*// Hiển thị danh sách Employee lên bảng*

public void displayEmployees(List<Employee> employees) {

tableModel.setRowCount(0);

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

for (Employee emp : employees) {

Object[] row = {

emp.getEmpId(),

emp.getFullName(),

emp.getGender(),

sdf.format(emp.getBirthDate()),

emp.getDepartment().getDeptName(),

emp.getSalary()

};

tableModel.addRow(row);

}

}

*// Hiển thị danh sách Department vào JComboBox*

public void populateDepartments(List<Department> departments) {

cbDepartment.removeAllItems();

for (Department dept : departments) {

cbDepartment.addItem(dept);

}

}

}

**4.5. Controller (EmployeeController.java)**

Điều khiển tương tác giữa View và Model.

package com.example.controller;

import com.example.dao.DepartmentDAO;

import com.example.dao.EmployeeDAO;

import com.example.model.Department;

import com.example.model.Employee;

import com.example.view.EmployeeView;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

public class **EmployeeController** {

private EmployeeView view;

private EmployeeDAO employeeDAO;

private DepartmentDAO departmentDAO;

public EmployeeController(EmployeeView view, EmployeeDAO employeeDAO, DepartmentDAO departmentDAO) {

this.view = view;

this.employeeDAO = employeeDAO;

this.departmentDAO = departmentDAO;

*// Load danh sách Department vào JComboBox*

try {

List<Department> departments = departmentDAO.getAllDepartments();

view.populateDepartments(departments);

} catch (Exception e) {

e.printStackTrace();

}

*// Thêm các sự kiện cho các nút*

view.getBtnAdd().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

Employee employee = createEmployeeFromInput();

employeeDAO.addEmployee(employee);

view.displayEmployees(employeeDAO.getAllEmployees());

clearFields();

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error adding employee: " + ex.getMessage());

}

}

});

view.getBtnUpdate().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

Employee employee = createEmployeeFromInput();

employeeDAO.updateEmployee(employee);

view.displayEmployees(employeeDAO.getAllEmployees());

clearFields();

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error updating employee: " + ex.getMessage());

}

}

});

view.getBtnDelete().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

int empId = Integer.parseInt(view.getTxtEmpId().getText());

employeeDAO.deleteEmployee(empId);

view.displayEmployees(employeeDAO.getAllEmployees());

clearFields();

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error deleting employee: " + ex.getMessage());

}

}

});

view.getBtnLoad().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

view.displayEmployees(employeeDAO.getAllEmployees());

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error loading employees: " + ex.getMessage());

}

}

});

view.getTable().getSelectionModel().addListSelectionListener(e -> {

int selectedRow = view.getTable().getSelectedRow();

if (selectedRow >= 0) {

view.getTxtEmpId().setText(view.getTable().getValueAt(selectedRow, 0).toString());

view.getTxtFullName().setText(view.getTable().getValueAt(selectedRow, 1).toString());

view.getTxtGender().setText(view.getTable().getValueAt(selectedRow, 2).toString());

view.getTxtBirthDate().setText(view.getTable().getValueAt(selectedRow, 3).toString());

view.getTxtSalary().setText(view.getTable().getValueAt(selectedRow, 5).toString());

String deptName = view.getTable().getValueAt(selectedRow, 4).toString();

for (int i = 0; i < view.getCbDepartment().getItemCount(); i++) {

if (view.getCbDepartment().getItemAt(i).getDeptName().equals(deptName)) {

view.getCbDepartment().setSelectedIndex(i);

break;

}

}

}

});

}

private Employee createEmployeeFromInput() throws Exception {

int empId = view.getTxtEmpId().getText().isEmpty() ? 0 : Integer.parseInt(view.getTxtEmpId().getText());

String fullName = view.getTxtFullName().getText();

String gender = view.getTxtGender().getText();

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

Date birthDate = sdf.parse(view.getTxtBirthDate().getText());

Department department = (Department) view.getCbDepartment().getSelectedItem();

double salary = Double.parseDouble(view.getTxtSalary().getText());

return new Employee(empId, fullName, gender, birthDate, department, salary);

}

private void clearFields() {

view.getTxtEmpId().setText("");

view.getTxtFullName().setText("");

view.getTxtGender().setText("");

view.getTxtBirthDate().setText("");

view.getTxtSalary().setText("");

view.getCbDepartment().setSelectedIndex(0);

}

}

**4.6. Main Class (Main.java)**

package com.example;

import com.example.controller.EmployeeController;

import com.example.dao.DepartmentDAO;

import com.example.dao.EmployeeDAO;

import com.example.view.EmployeeView;

public class **Main** {

}

**5. Chạy chương trình**

1. Đảm bảo MySQL server đang chạy và bạn đã tạo cơ sở dữ liệu EMS như ở bước 2.
2. Cập nhật thông tin USER và PASSWORD trong DatabaseConnection.java để khớp với MySQL của bạn.
3. Chạy file Main.java từ IntelliJ.

**6. Giải thích các thay đổi**

* **Thay JTextField thành JComboBox cho Department**: Trong EmployeeView, tôi đã thay JTextField của Department thành JComboBox<Department> để hiển thị danh sách các phòng ban từ bảng Department. Khi chọn một phòng ban, JComboBox sẽ trả về đối tượng Department tương ứng.
* **Tác vụ CRUD**:
  + **Add**: Thêm nhân viên mới với thông tin từ các trường nhập liệu, bao gồm phòng ban được chọn từ JComboBox.
  + **Update**: Cập nhật thông tin nhân viên, bao gồm phòng ban mới nếu thay đổi.
  + **Delete**: Xóa nhân viên dựa trên empId.
  + **Load**: Tải lại toàn bộ danh sách nhân viên lên bảng.
* **Tìm kiếm**: các chức năng tìm kiếm theo Department và Salary, nhưng không tích hợp giao diện tìm kiếm trong mã này (có thể mở rộng nếu cần).

**7. Hạn chế và cải tiến**

* **Xử lý lỗi**: Hiện tại, chương trình chưa xử lý triệt để các trường hợp nhập liệu sai (ví dụ: định dạng ngày không đúng). Bạn có thể thêm kiểm tra dữ liệu đầu vào.
* **Tìm kiếm**: Chưa có giao diện cho tìm kiếm theo Department và Salary. Bạn có thể thêm JComboBox và JTextField để hỗ trợ tìm kiếm.
* **Giao diện**: Giao diện có thể được cải thiện bằng cách sử dụng Layout Manager khác hoặc thêm các thành phần như JLabel để hiển thị kết quả tìm kiếm.

A screenshot of a computer

AI-generated content may be incorrect.

Dựa trên phần hạn chế và cải tiến đã đề cập trong mã trước, thực hiện cải thiện chương trình bằng cách:

1. **Xử lý lỗi nhập liệu**: Thêm kiểm tra dữ liệu đầu vào để đảm bảo định dạng ngày hợp lệ, các trường không để trống, và giá trị salary phải là số dương.
2. **Thêm giao diện tìm kiếm**: Tích hợp giao diện tìm kiếm theo Department và Salary bằng cách thêm JComboBox và JTextField tương ứng.
3. **Cải thiện giao diện**: Sử dụng GridBagLayout để bố cục giao diện rõ ràng hơn và thêm các thông báo lỗi hoặc thành công.

Dưới đây là mã đã được cải tiến, tập trung vào các phần này. Sẽ có chỉnh sửa các file liên quan và giải thích chi tiết.

**1. Cải thiện xử lý lỗi nhập liệu**

Chúng ta sẽ thêm kiểm tra dữ liệu trong **EmployeeController** trước khi thực hiện các tác vụ như **Add** hoặc **Update**. Cụ thể:

* Kiểm tra các trường bắt buộc không được để trống.
* Kiểm tra định dạng ngày phải đúng (yyyy-MM-dd).
* Kiểm tra **salary** phải là số dương.

**Cập nhật EmployeeController.java**

Thêm phương thức **validateInput** để kiểm tra dữ liệu trước khi tạo **Employee**. Đồng thời hiển thị thông báo lỗi nếu có vấn đề.

package com.example.controller;

import com.example.dao.DepartmentDAO;

import com.example.dao.EmployeeDAO;

import com.example.model.Department;

import com.example.model.Employee;

import com.example.view.EmployeeView;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

public class **EmployeeController** {

private EmployeeView view;

private EmployeeDAO employeeDAO;

private DepartmentDAO departmentDAO;

public EmployeeController(EmployeeView view, EmployeeDAO employeeDAO, DepartmentDAO departmentDAO) {

this.view = view;

this.employeeDAO = employeeDAO;

this.departmentDAO = departmentDAO;

*// Load danh sách Department vào JComboBox*

try {

List<Department> departments = departmentDAO.getAllDepartments();

view.populateDepartments(departments);

view.getCbSearchDepartment().setModel(new DefaultComboBoxModel<>(departments.toArray(new Department[0])));

} catch (Exception e) {

JOptionPane.showMessageDialog(view, "Error loading departments: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

*// Thêm sự kiện cho các nút*

view.getBtnAdd().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

if (!validateInput()) return;

Employee employee = createEmployeeFromInput();

employeeDAO.addEmployee(employee);

view.displayEmployees(employeeDAO.getAllEmployees());

clearFields();

JOptionPane.showMessageDialog(view, "Employee added successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error adding employee: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

view.getBtnUpdate().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

if (!validateInput()) return;

Employee employee = createEmployeeFromInput();

employeeDAO.updateEmployee(employee);

view.displayEmployees(employeeDAO.getAllEmployees());

clearFields();

JOptionPane.showMessageDialog(view, "Employee updated successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error updating employee: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

view.getBtnDelete().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

if (view.getTxtEmpId().getText().isEmpty()) {

JOptionPane.showMessageDialog(view, "Please select an employee to delete!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int empId = Integer.parseInt(view.getTxtEmpId().getText());

employeeDAO.deleteEmployee(empId);

view.displayEmployees(employeeDAO.getAllEmployees());

clearFields();

JOptionPane.showMessageDialog(view, "Employee deleted successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error deleting employee: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

view.getBtnLoad().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

view.displayEmployees(employeeDAO.getAllEmployees());

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error loading employees: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

*// Sự kiện tìm kiếm theo Department*

view.getBtnSearchByDepartment().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

Department selectedDept = (Department) view.getCbSearchDepartment().getSelectedItem();

if (selectedDept != null) {

List<Employee> employees = employeeDAO.searchByDepartment(selectedDept.getDeptId());

view.displayEmployees(employees);

}

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error searching by department: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

*// Sự kiện tìm kiếm theo Salary*

view.getBtnSearchBySalary().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

String salaryText = view.getTxtSearchSalary().getText();

if (salaryText.isEmpty()) {

JOptionPane.showMessageDialog(view, "Please enter a salary to search!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

double salary = Double.parseDouble(salaryText);

if (salary <= 0) {

JOptionPane.showMessageDialog(view, "Salary must be a positive number!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

List<Employee> employees = employeeDAO.searchBySalary(salary);

view.displayEmployees(employees);

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(view, "Salary must be a valid number!", "Error", JOptionPane.ERROR\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error searching by salary: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

view.getTable().getSelectionModel().addListSelectionListener(e -> {

int selectedRow = view.getTable().getSelectedRow();

if (selectedRow >= 0) {

view.getTxtEmpId().setText(view.getTable().getValueAt(selectedRow, 0).toString());

view.getTxtFullName().setText(view.getTable().getValueAt(selectedRow, 1).toString());

view.getTxtGender().setText(view.getTable().getValueAt(selectedRow, 2).toString());

view.getTxtBirthDate().setText(view.getTable().getValueAt(selectedRow, 3).toString());

view.getTxtSalary().setText(view.getTable().getValueAt(selectedRow, 5).toString());

String deptName = view.getTable().getValueAt(selectedRow, 4).toString();

for (int i = 0; i < view.getCbDepartment().getItemCount(); i++) {

if (view.getCbDepartment().getItemAt(i).getDeptName().equals(deptName)) {

view.getCbDepartment().setSelectedIndex(i);

break;

}

}

}

});

}

private Employee createEmployeeFromInput() throws Exception {

int empId = view.getTxtEmpId().getText().isEmpty() ? 0 : Integer.parseInt(view.getTxtEmpId().getText());

String fullName = view.getTxtFullName().getText();

String gender = view.getTxtGender().getText();

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

Date birthDate = sdf.parse(view.getTxtBirthDate().getText());

Department department = (Department) view.getCbDepartment().getSelectedItem();

double salary = Double.parseDouble(view.getTxtSalary().getText());

return new Employee(empId, fullName, gender, birthDate, department, salary);

}

private boolean validateInput() {

*// Kiểm tra trường trống*

if (view.getTxtFullName().getText().isEmpty() ||

view.getTxtGender().getText().isEmpty() ||

view.getTxtBirthDate().getText().isEmpty() ||

view.getTxtSalary().getText().isEmpty()) {

JOptionPane.showMessageDialog(view, "All fields must be filled!", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

*// Kiểm tra định dạng ngày*

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

sdf.setLenient(false);

try {

sdf.parse(view.getTxtBirthDate().getText());

} catch (Exception e) {

JOptionPane.showMessageDialog(view, "Birth date must be in format yyyy-MM-dd!", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

*// Kiểm tra salary*

try {

double salary = Double.parseDouble(view.getTxtSalary().getText());

if (salary <= 0) {

JOptionPane.showMessageDialog(view, "Salary must be a positive number!", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(view, "Salary must be a valid number!", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

private void clearFields() {

view.getTxtEmpId().setText("");

view.getTxtFullName().setText("");

view.getTxtGender().setText("");

view.getTxtBirthDate().setText("");

view.getTxtSalary().setText("");

view.getCbDepartment().setSelectedIndex(0);

view.getTxtSearchSalary().setText("");

}

}

**2. Thêm giao diện tìm kiếm**

Chúng ta sẽ cập nhật **EmployeeView** để thêm hai phần tìm kiếm:

* Tìm kiếm theo **Department** với **JComboBox**.
* Tìm kiếm theo **Salary** với **JTextField**.

**Cập nhật EmployeeView.java**

Sử dụng GridBagLayout để bố cục giao diện hợp lý hơn. Thêm các thành phần tìm kiếm và nút bấm tương ứng.

package com.example.view;

import com.example.model.Department;

import com.example.model.Employee;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.text.SimpleDateFormat;

import java.util.List;

public class **EmployeeView** extends **JFrame** {

private JTextField txtEmpId, txtFullName, txtGender, txtBirthDate, txtSalary, txtSearchSalary;

private JComboBox<Department> cbDepartment, cbSearchDepartment;

private JButton btnAdd, btnUpdate, btnDelete, btnLoad, btnSearchByDepartment, btnSearchBySalary;

private JTable table;

private DefaultTableModel tableModel;

public EmployeeView() {

setTitle("EMPLOYEE MANAGEMENT");

setSize(800, 600);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

*// Panel chính sử dụng GridBagLayout*

JPanel mainPanel = new JPanel(new GridBagLayout());

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(5, 5, 5, 5); *// Khoảng cách giữa các thành phần*

gbc.fill = GridBagConstraints.HORIZONTAL;

*// Panel nhập liệu*

JPanel inputPanel = new JPanel(new GridLayout(6, 2, 5, 5));

inputPanel.setBorder(BorderFactory.createTitledBorder("Employee Information"));

inputPanel.add(new JLabel("EMPLOYEE ID:"));

txtEmpId = new JTextField();

txtEmpId.setEditable(false);

inputPanel.add(txtEmpId);

inputPanel.add(new JLabel("FULL NAME:"));

txtFullName = new JTextField();

inputPanel.add(txtFullName);

inputPanel.add(new JLabel("GENDER:"));

txtGender = new JTextField();

inputPanel.add(txtGender);

inputPanel.add(new JLabel("BIRTH DATE:"));

txtBirthDate = new JTextField();

inputPanel.add(txtBirthDate);

inputPanel.add(new JLabel("DEPARTMENT:"));

cbDepartment = new JComboBox<>();

inputPanel.add(cbDepartment);

inputPanel.add(new JLabel("SALARY:"));

txtSalary = new JTextField();

inputPanel.add(txtSalary);

gbc.gridx = 0;

gbc.gridy = 0;

gbc.gridwidth = 2;

mainPanel.add(inputPanel, gbc);

*// Panel nút bấm*

JPanel buttonPanel = new JPanel(new FlowLayout());

btnAdd = new JButton("ADD");

btnUpdate = new JButton("UPDATE");

btnDelete = new JButton("DELETE");

btnLoad = new JButton("LOAD");

buttonPanel.add(btnAdd);

buttonPanel.add(btnUpdate);

buttonPanel.add(btnDelete);

buttonPanel.add(btnLoad);

gbc.gridx = 0;

gbc.gridy = 1;

gbc.gridwidth = 2;

mainPanel.add(buttonPanel, gbc);

*// Panel tìm kiếm*

JPanel searchPanel = new JPanel(new GridLayout(2, 3, 5, 5));

searchPanel.setBorder(BorderFactory.createTitledBorder("Search Options"));

searchPanel.add(new JLabel("SEARCH BY DEPARTMENT:"));

cbSearchDepartment = new JComboBox<>();

searchPanel.add(cbSearchDepartment);

btnSearchByDepartment = new JButton("SEARCH");

searchPanel.add(btnSearchByDepartment);

searchPanel.add(new JLabel("SEARCH BY SALARY:"));

txtSearchSalary = new JTextField();

searchPanel.add(txtSearchSalary);

btnSearchBySalary = new JButton("SEARCH");

searchPanel.add(btnSearchBySalary);

gbc.gridx = 0;

gbc.gridy = 2;

gbc.gridwidth = 2;

mainPanel.add(searchPanel, gbc);

*// Bảng hiển thị dữ liệu*

String[] columns = {"ID", "FULL NAME", "GENDER", "BIRTH DATE", "DEPARTMENT", "SALARY"};

tableModel = new DefaultTableModel(columns, 0);

table = new JTable(tableModel);

JScrollPane scrollPane = new JScrollPane(table);

gbc.gridx = 0;

gbc.gridy = 3;

gbc.gridwidth = 2;

gbc.weightx = 1.0;

gbc.weighty = 1.0;

gbc.fill = GridBagConstraints.BOTH;

mainPanel.add(scrollPane, gbc);

*// Thêm mainPanel vào JFrame*

add(mainPanel, BorderLayout.CENTER);

}

*// Getters cho Controller*

public JTextField getTxtEmpId() {

return txtEmpId;

}

public JTextField getTxtFullName() {

return txtFullName;

}

public JTextField getTxtGender() {

return txtGender;

}

public JTextField getTxtBirthDate() {

return txtBirthDate;

}

public JComboBox<Department> getCbDepartment() {

return cbDepartment;

}

public JTextField getTxtSalary() {

return txtSalary;

}

public JComboBox<Department> getCbSearchDepartment() {

return cbSearchDepartment;

}

public JTextField getTxtSearchSalary() {

return txtSearchSalary;

}

public JButton getBtnAdd() {

return btnAdd;

}

public JButton getBtnUpdate() {

return btnUpdate;

}

public JButton getBtnDelete() {

return btnDelete;

}

public JButton getBtnLoad() {

return btnLoad;

}

public JButton getBtnSearchByDepartment() {

return btnSearchByDepartment;

}

public JButton getBtnSearchBySalary() {

return btnSearchBySalary;

}

public JTable getTable() {

return table;

}

public DefaultTableModel getTableModel() {

return tableModel;

}

*// Hiển thị danh sách Employee lên bảng*

public void displayEmployees(List<Employee> employees) {

tableModel.setRowCount(0);

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

for (Employee emp : employees) {

Object[] row = {

emp.getEmpId(),

emp.getFullName(),

emp.getGender(),

sdf.format(emp.getBirthDate()),

emp.getDepartment().getDeptName(),

emp.getSalary()

};

tableModel.addRow(row);

}

}

*// Hiển thị danh sách Department vào JComboBox*

public void populateDepartments(List<Department> departments) {

cbDepartment.removeAllItems();

for (Department dept : departments) {

cbDepartment.addItem(dept);

}

}

}

**3. Cải thiện giao diện**

* **Sử dụng GridBagLayout**: Trong **EmployeeView**, đã thay đổi bố cục từ **BorderLayout** đơn giản sang **GridBagLayout** để sắp xếp các thành phần (**input panel, button panel, search panel**, và bảng dữ liệu) một cách linh hoạt hơn.
* **Thêm viền và tiêu đề**: Sử dụng **BorderFactory**.**createTitledBorder** để thêm tiêu đề cho các panel, giúp giao diện rõ ràng hơn.
* **Thông báo người dùng**: Trong **EmployeeController**, đã thêm các thông báo thành công hoặc lỗi bằng **JOptionPane** để cải thiện trải nghiệm người dùng.

**4. Tổng hợp các thay đổi**

**Xử lý lỗi nhập liệu**

* Thêm **validateInput** trong **EmployeeController** để kiểm tra:
  + Các trường không được để trống.
  + Định dạng ngày phải đúng (yyyy-MM-dd).
  + **Salary** phải là số dương.
* Hiển thị thông báo lỗi cụ thể bằng **JOptionPane**.

**Thêm giao diện tìm kiếm**

* Trong **EmployeeView**, thêm **panel** tìm kiếm với:
  + **JComboBox** để chọn **Department** cần tìm.
  + **JTextField** để nhập **Salary** cần tìm.
  + Hai nút **SEARCH** tương ứng.
* Trong **EmployeeController**, xử lý sự kiện tìm kiếm:
  + Tìm kiếm theo **Department** bằng cách gọi **employeeDAO**.**searchByDepartment**.
  + Tìm kiếm theo **Salary** bằng cách gọi **employeeDAO**.**searchBySalary**, với kiểm tra đầu vào.

**Cải thiện giao diện**

* Sử dụng **GridBagLayout** để bố cục các thành phần hợp lý hơn.
* Thêm viền và tiêu đề cho các **panel**.
* Hiển thị thông báo thành công hoặc lỗi sau mỗi tác vụ (**Add**, **Update**, **Delete**).

**5. Chạy chương trình**

1. Đảm bảo bạn đã thiết lập cơ sở dữ liệu **EMS** như trong phần trước.
2. Cập nhật thông tin **USER** và **PASSWORD** trong **DatabaseConnection**.java.
3. Chạy file **Main**.java từ **IntelliJ**.

**6. Kết quả mong đợi**

* **Giao diện**: Giao diện sẽ được chia thành các phần rõ ràng:
  + Phần nhập liệu ở trên cùng.
  + Phần nút bấm (**ADD, UPDATE, DELETE, LOAD**) ở giữa.
  + Phần tìm kiếm (**SEARCH BY DEPARTMENT, SEARCH BY SALARY**) ngay dưới.
  + Bảng dữ liệu ở dưới cùng.
* **Xử lý lỗi**: Nếu người dùng nhập sai (ví dụ: để trống trường, ngày không đúng định dạng, hoặc salary âm), chương trình sẽ hiển thị thông báo lỗi.
* **Tìm kiếm**: Người dùng có thể tìm kiếm nhân viên theo **Department** hoặc **Salary** thông qua các thành phần giao diện mới.

**7. Mở rộng thêm (nếu cần)**

* **Thêm nút Clear**: Có thể thêm một nút "**CLEAR**" để xóa các trường nhập liệu và ô tìm kiếm.
* **Tìm kiếm nâng cao**: Thêm khả năng tìm kiếm theo khoảng lương (ví dụ: từ X đến Y) thay vì chỉ tìm kiếm chính xác.
* **Giao diện đẹp hơn**: Sử dụng thư viện giao diện như **FlatLaf** hoặc thêm màu sắc, biểu tượng cho các nút.

Mở rộng chương trình theo các mục đã đề cập trong phần **Mở rộng thêm**. Cụ thể như sau:

1. **Thêm nút Clear**: Thêm một nút "CLEAR" để xóa các trường nhập liệu và ô tìm kiếm.
2. **Tìm kiếm nâng cao**: Thêm khả năng tìm kiếm theo khoảng lương (ví dụ: từ X đến Y) thay vì chỉ tìm kiếm chính xác.
3. **Cải thiện giao diện**: Sử dụng thư viện giao diện **FlatLaf** để làm cho giao diện đẹp hơn và thêm màu sắc, biểu tượng cho các nút.

Dưới đây là mã nguồn đã được cập nhật và giải thích chi tiết.

**1. Thêm nút Clear**

Nút "**CLEAR**" sẽ xóa tất cả các trường nhập liệu (ID, Full Name, Gender, Birth Date, Salary) và ô tìm kiếm (Salary Range). Sẽ cập nhật EmployeeView để thêm nút này và EmployeeController để xử lý sự kiện.

**Cập nhật EmployeeView.java**

Thêm nút "**CLEAR**" vào panel chứa các nút bấm.

package com.example.view;

import com.example.model.Department;

import com.example.model.Employee;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.text.SimpleDateFormat;

import java.util.List;

public class **EmployeeView** extends **JFrame** {

private JTextField txtEmpId, txtFullName, txtGender, txtBirthDate, txtSalary, txtSearchSalaryFrom, txtSearchSalaryTo;

private JComboBox<Department> cbDepartment, cbSearchDepartment;

private JButton btnAdd, btnUpdate, btnDelete, btnLoad, btnSearchByDepartment, btnSearchBySalary, btnClear;

private JTable table;

private DefaultTableModel tableModel;

public EmployeeView() {

setTitle("EMPLOYEE MANAGEMENT");

setSize(800, 600);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

*// Panel chính sử dụng GridBagLayout*

JPanel mainPanel = new JPanel(new GridBagLayout());

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(5, 5, 5, 5); *// Khoảng cách giữa các thành phần*

gbc.fill = GridBagConstraints.HORIZONTAL;

*// Panel nhập liệu*

JPanel inputPanel = new JPanel(new GridLayout(6, 2, 5, 5));

inputPanel.setBorder(BorderFactory.createTitledBorder("Employee Information"));

inputPanel.add(new JLabel("EMPLOYEE ID:"));

txtEmpId = new JTextField();

txtEmpId.setEditable(false);

inputPanel.add(txtEmpId);

inputPanel.add(new JLabel("FULL NAME:"));

txtFullName = new JTextField();

inputPanel.add(txtFullName);

inputPanel.add(new JLabel("GENDER:"));

txtGender = new JTextField();

inputPanel.add(txtGender);

inputPanel.add(new JLabel("BIRTH DATE:"));

txtBirthDate = new JTextField();

inputPanel.add(txtBirthDate);

inputPanel.add(new JLabel("DEPARTMENT:"));

cbDepartment = new JComboBox<>();

inputPanel.add(cbDepartment);

inputPanel.add(new JLabel("SALARY:"));

txtSalary = new JTextField();

inputPanel.add(txtSalary);

gbc.gridx = 0;

gbc.gridy = 0;

gbc.gridwidth = 2;

mainPanel.add(inputPanel, gbc);

*// Panel nút bấm*

JPanel buttonPanel = new JPanel(new FlowLayout());

btnAdd = new JButton("ADD");

btnAdd.setIcon(new ImageIcon(getClass().getResource("/icons/add.png"))); *// Thêm biểu tượng (tùy chọn)*

btnUpdate = new JButton("UPDATE");

btnUpdate.setIcon(new ImageIcon(getClass().getResource("/icons/update.png")));

btnDelete = new JButton("DELETE");

btnDelete.setIcon(new ImageIcon(getClass().getResource("/icons/delete.png")));

btnLoad = new JButton("LOAD");

btnLoad.setIcon(new ImageIcon(getClass().getResource("/icons/load.png")));

btnClear = new JButton("CLEAR");

btnClear.setIcon(new ImageIcon(getClass().getResource("/icons/clear.png")));

buttonPanel.add(btnAdd);

buttonPanel.add(btnUpdate);

buttonPanel.add(btnDelete);

buttonPanel.add(btnLoad);

buttonPanel.add(btnClear);

gbc.gridx = 0;

gbc.gridy = 1;

gbc.gridwidth = 2;

mainPanel.add(buttonPanel, gbc);

*// Panel tìm kiếm*

JPanel searchPanel = new JPanel(new GridLayout(2, 4, 5, 5));

searchPanel.setBorder(BorderFactory.createTitledBorder("Search Options"));

searchPanel.add(new JLabel("SEARCH BY DEPARTMENT:"));

cbSearchDepartment = new JComboBox<>();

searchPanel.add(cbSearchDepartment);

btnSearchByDepartment = new JButton("SEARCH");

btnSearchByDepartment.setIcon(new ImageIcon(getClass().getResource("/icons/search.png")));

searchPanel.add(btnSearchByDepartment);

searchPanel.add(new JLabel()); *// Placeholder*

searchPanel.add(new JLabel("SALARY FROM:"));

txtSearchSalaryFrom = new JTextField();

searchPanel.add(txtSearchSalaryFrom);

searchPanel.add(new JLabel("SALARY TO:"));

txtSearchSalaryTo = new JTextField();

searchPanel.add(txtSearchSalaryTo);

btnSearchBySalary = new JButton("SEARCH");

btnSearchBySalary.setIcon(new ImageIcon(getClass().getResource("/icons/search.png")));

searchPanel.add(btnSearchBySalary);

gbc.gridx = 0;

gbc.gridy = 2;

gbc.gridwidth = 2;

mainPanel.add(searchPanel, gbc);

*// Bảng hiển thị dữ liệu*

String[] columns = {"ID", "FULL NAME", "GENDER", "BIRTH DATE", "DEPARTMENT", "SALARY"};

tableModel = new DefaultTableModel(columns, 0);

table = new JTable(tableModel);

JScrollPane scrollPane = new JScrollPane(table);

gbc.gridx = 0;

gbc.gridy = 3;

gbc.gridwidth = 2;

gbc.weightx = 1.0;

gbc.weighty = 1.0;

gbc.fill = GridBagConstraints.BOTH;

mainPanel.add(scrollPane, gbc);

*// Thêm mainPanel vào JFrame*

add(mainPanel, BorderLayout.CENTER);

}

*// Getters cho Controller*

public JTextField getTxtEmpId() {

return txtEmpId;

}

public JTextField getTxtFullName() {

return txtFullName;

}

public JTextField getTxtGender() {

return txtGender;

}

public JTextField getTxtBirthDate() {

return txtBirthDate;

}

public JComboBox<Department> getCbDepartment() {

return cbDepartment;

}

public JTextField getTxtSalary() {

return txtSalary;

}

public JComboBox<Department> getCbSearchDepartment() {

return cbSearchDepartment;

}

public JTextField getTxtSearchSalaryFrom() {

return txtSearchSalaryFrom;

}

public JTextField getTxtSearchSalaryTo() {

return txtSearchSalaryTo;

}

public JButton getBtnAdd() {

return btnAdd;

}

public JButton getBtnUpdate() {

return btnUpdate;

}

public JButton getBtnDelete() {

return btnDelete;

}

public JButton getBtnLoad() {

return btnLoad;

}

public JButton getBtnClear() {

return btnClear;

}

public JButton getBtnSearchByDepartment() {

return btnSearchByDepartment;

}

public JButton getBtnSearchBySalary() {

return btnSearchBySalary;

}

public JTable getTable() {

return table;

}

public DefaultTableModel getTableModel() {

return tableModel;

}

*// Hiển thị danh sách Employee lên bảng*

public void displayEmployees(List<Employee> employees) {

tableModel.setRowCount(0);

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

for (Employee emp : employees) {

Object[] row = {

emp.getEmpId(),

emp.getFullName(),

emp.getGender(),

sdf.format(emp.getBirthDate()),

emp.getDepartment().getDeptName(),

emp.getSalary()

};

tableModel.addRow(row);

}

}

*// Hiển thị danh sách Department vào JComboBox*

public void populateDepartments(List<Department> departments) {

cbDepartment.removeAllItems();

for (Department dept : departments) {

cbDepartment.addItem(dept);

}

}

}

**Cập nhật EmployeeController.java**

Thêm sự kiện cho nút "CLEAR" để xóa các trường nhập liệu và ô tìm kiếm.

package com.example.controller;

import com.example.dao.DepartmentDAO;

import com.example.dao.EmployeeDAO;

import com.example.model.Department;

import com.example.model.Employee;

import com.example.view.EmployeeView;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

public class **EmployeeController** {

private EmployeeView view;

private EmployeeDAO employeeDAO;

private DepartmentDAO departmentDAO;

public EmployeeController(EmployeeView view, EmployeeDAO employeeDAO, DepartmentDAO departmentDAO) {

this.view = view;

this.employeeDAO = employeeDAO;

this.departmentDAO = departmentDAO;

*// Load danh sách Department vào JComboBox*

try {

List<Department> departments = departmentDAO.getAllDepartments();

view.populateDepartments(departments);

view.getCbSearchDepartment().setModel(new DefaultComboBoxModel<>(departments.toArray(new Department[0])));

} catch (Exception e) {

JOptionPane.showMessageDialog(view, "Error loading departments: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

*// Thêm sự kiện cho các nút*

view.getBtnAdd().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

if (!validateInput()) return;

Employee employee = createEmployeeFromInput();

employeeDAO.addEmployee(employee);

view.displayEmployees(employeeDAO.getAllEmployees());

clearFields();

JOptionPane.showMessageDialog(view, "Employee added successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error adding employee: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

view.getBtnUpdate().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

if (!validateInput()) return;

Employee employee = createEmployeeFromInput();

employeeDAO.updateEmployee(employee);

view.displayEmployees(employeeDAO.getAllEmployees());

clearFields();

JOptionPane.showMessageDialog(view, "Employee updated successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error updating employee: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

view.getBtnDelete().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

if (view.getTxtEmpId().getText().isEmpty()) {

JOptionPane.showMessageDialog(view, "Please select an employee to delete!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int empId = Integer.parseInt(view.getTxtEmpId().getText());

employeeDAO.deleteEmployee(empId);

view.displayEmployees(employeeDAO.getAllEmployees());

clearFields();

JOptionPane.showMessageDialog(view, "Employee deleted successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error deleting employee: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

view.getBtnLoad().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

view.displayEmployees(employeeDAO.getAllEmployees());

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error loading employees: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

view.getBtnClear().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

clearFields();

JOptionPane.showMessageDialog(view, "Fields cleared!", "Info", JOptionPane.INFORMATION\_MESSAGE);

}

});

*// Sự kiện tìm kiếm theo Department*

view.getBtnSearchByDepartment().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

Department selectedDept = (Department) view.getCbSearchDepartment().getSelectedItem();

if (selectedDept != null) {

List<Employee> employees = employeeDAO.searchByDepartment(selectedDept.getDeptId());

view.displayEmployees(employees);

}

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error searching by department: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

*// Sự kiện tìm kiếm theo Salary Range*

view.getBtnSearchBySalary().addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

String salaryFromText = view.getTxtSearchSalaryFrom().getText();

String salaryToText = view.getTxtSearchSalaryTo().getText();

if (salaryFromText.isEmpty() || salaryToText.isEmpty()) {

JOptionPane.showMessageDialog(view, "Please enter both Salary From and Salary To!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

double salaryFrom = Double.parseDouble(salaryFromText);

double salaryTo = Double.parseDouble(salaryToText);

if (salaryFrom < 0 || salaryTo < 0) {

JOptionPane.showMessageDialog(view, "Salary values must be positive!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

if (salaryFrom > salaryTo) {

JOptionPane.showMessageDialog(view, "Salary From must be less than or equal to Salary To!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

List<Employee> employees = employeeDAO.searchBySalaryRange(salaryFrom, salaryTo);

view.displayEmployees(employees);

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(view, "Salary values must be valid numbers!", "Error", JOptionPane.ERROR\_MESSAGE);

} catch (Exception ex) {

JOptionPane.showMessageDialog(view, "Error searching by salary range: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

view.getTable().getSelectionModel().addListSelectionListener(e -> {

int selectedRow = view.getTable().getSelectedRow();

if (selectedRow >= 0) {

view.getTxtEmpId().setText(view.getTable().getValueAt(selectedRow, 0).toString());

view.getTxtFullName().setText(view.getTable().getValueAt(selectedRow, 1).toString());

view.getTxtGender().setText(view.getTable().getValueAt(selectedRow, 2).toString());

view.getTxtBirthDate().setText(view.getTable().getValueAt(selectedRow, 3).toString());

view.getTxtSalary().setText(view.getTable().getValueAt(selectedRow, 5).toString());

String deptName = view.getTable().getValueAt(selectedRow, 4).toString();

for (int i = 0; i < view.getCbDepartment().getItemCount(); i++) {

if (view.getCbDepartment().getItemAt(i).getDeptName().equals(deptName)) {

view.getCbDepartment().setSelectedIndex(i);

break;

}

}

}

});

}

private Employee createEmployeeFromInput() throws Exception {

int empId = view.getTxtEmpId().getText().isEmpty() ? 0 : Integer.parseInt(view.getTxtEmpId().getText());

String fullName = view.getTxtFullName().getText();

String gender = view.getTxtGender().getText();

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

Date birthDate = sdf.parse(view.getTxtBirthDate().getText());

Department department = (Department) view.getCbDepartment().getSelectedItem();

double salary = Double.parseDouble(view.getTxtSalary().getText());

return new Employee(empId, fullName, gender, birthDate, department, salary);

}

private boolean validateInput() {

*// Kiểm tra trường trống*

if (view.getTxtFullName().getText().isEmpty() ||

view.getTxtGender().getText().isEmpty() ||

view.getTxtBirthDate().getText().isEmpty() ||

view.getTxtSalary().getText().isEmpty()) {

JOptionPane.showMessageDialog(view, "All fields must be filled!", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

*// Kiểm tra định dạng ngày*

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

sdf.setLenient(false);

try {

sdf.parse(view.getTxtBirthDate().getText());

} catch (Exception e) {

JOptionPane.showMessageDialog(view, "Birth date must be in format yyyy-MM-dd!", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

*// Kiểm tra salary*

try {

double salary = Double.parseDouble(view.getTxtSalary().getText());

if (salary <= 0) {

JOptionPane.showMessageDialog(view, "Salary must be a positive number!", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(view, "Salary must be a valid number!", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

private void clearFields() {

view.getTxtEmpId().setText("");

view.getTxtFullName().setText("");

view.getTxtGender().setText("");

view.getTxtBirthDate().setText("");

view.getTxtSalary().setText("");

view.getCbDepartment().setSelectedIndex(0);

view.getTxtSearchSalaryFrom().setText("");

view.getTxtSearchSalaryTo().setText("");

}

}

**2. Tìm kiếm nâng cao (theo khoảng lương)**

Để hỗ trợ tìm kiếm theo khoảng lương (Salary From và Salary To), chúng ta cần:

* Thay đổi giao diện tìm kiếm trong EmployeeView để có hai ô nhập liệu: Salary From và Salary To.
* Cập nhật EmployeeDAO để thêm phương thức searchBySalaryRange.
* Cập nhật EmployeeController để xử lý tìm kiếm theo khoảng lương.

**Đã cập nhật EmployeeView.java**

Trong phần trước, thay đổi giao diện tìm kiếm từ một ô Salary thành hai ô Salary From và Salary To. Xem phần searchPanel trong EmployeeView.java.

**Cập nhật EmployeeDAO.java**

Thêm phương thức searchBySalaryRange để tìm kiếm nhân viên trong khoảng lương.

package com.example.dao;

import com.example.model.Department;

import com.example.model.Employee;

import com.example.util.DatabaseConnection;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class **EmployeeDAO** {

private final DepartmentDAO departmentDAO = new DepartmentDAO();

public void addEmployee(Employee employee) throws SQLException {

String query = "INSERT INTO Employee (fullName, gender, birthDate, deptId, salary) VALUES (?, ?, ?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, employee.getFullName());

stmt.setString(2, employee.getGender());

stmt.setDate(3, new java.sql.Date(employee.getBirthDate().getTime()));

stmt.setInt(4, employee.getDepartment().getDeptId());

stmt.setDouble(5, employee.getSalary());

stmt.executeUpdate();

}

}

public void updateEmployee(Employee employee) throws SQLException {

String query = "UPDATE Employee SET fullName = ?, gender = ?, birthDate = ?, deptId = ?, salary = ? WHERE empId = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, employee.getFullName());

stmt.setString(2, employee.getGender());

stmt.setDate(3, new java.sql.Date(employee.getBirthDate().getTime()));

stmt.setInt(4, employee.getDepartment().getDeptId());

stmt.setDouble(5, employee.getSalary());

stmt.setInt(6, employee.getEmpId());

stmt.executeUpdate();

}

}

public void deleteEmployee(int empId) throws SQLException {

String query = "DELETE FROM Employee WHERE empId = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, empId);

stmt.executeUpdate();

}

}

public List<Employee> getAllEmployees() throws SQLException {

List<Employee> employees = new ArrayList<>();

String query = "SELECT \* FROM Employee";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Department dept = departmentDAO.getDepartmentById(rs.getInt("deptId"));

Employee emp = new Employee(

rs.getInt("empId"),

rs.getString("fullName"),

rs.getString("gender"),

rs.getDate("birthDate"),

dept,

rs.getDouble("salary")

);

employees.add(emp);

}

}

return employees;

}

public List<Employee> searchByDepartment(int deptId) throws SQLException {

List<Employee> employees = new ArrayList<>();

String query = "SELECT \* FROM Employee WHERE deptId = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, deptId);

try (ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Department dept = departmentDAO.getDepartmentById(rs.getInt("deptId"));

Employee emp = new Employee(

rs.getInt("empId"),

rs.getString("fullName"),

rs.getString("gender"),

rs.getDate("birthDate"),

dept,

rs.getDouble("salary")

);

employees.add(emp);

}

}

}

return employees;

}

public List<Employee> searchBySalary(double salary) throws SQLException {

List<Employee> employees = new ArrayList<>();

String query = "SELECT \* FROM Employee WHERE salary = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setDouble(1, salary);

try (ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Department dept = departmentDAO.getDepartmentById(rs.getInt("deptId"));

Employee emp = new Employee(

rs.getInt("empId"),

rs.getString("fullName"),

rs.getString("gender"),

rs.getDate("birthDate"),

dept,

rs.getDouble("salary")

);

employees.add(emp);

}

}

}

return employees;

}

public List<Employee> searchBySalaryRange(double salaryFrom, double salaryTo) throws SQLException {

List<Employee> employees = new ArrayList<>();

String query = "SELECT \* FROM Employee WHERE salary BETWEEN ? AND ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setDouble(1, salaryFrom);

stmt.setDouble(2, salaryTo);

try (ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Department dept = departmentDAO.getDepartmentById(rs.getInt("deptId"));

Employee emp = new Employee(

rs.getInt("empId"),

rs.getString("fullName"),

rs.getString("gender"),

rs.getDate("birthDate"),

dept,

rs.getDouble("salary")

);

employees.add(emp);

}

}

}

return employees;

}

}

**Đã cập nhật EmployeeController.java**

Trong EmployeeController, đã cập nhật sự kiện của nút "SEARCH" (theo Salary) để xử lý tìm kiếm theo khoảng lương. Xem phần xử lý sự kiện btnSearchBySalary.

**3. Cải thiện giao diện với FlatLaf**

**FlatLaf** là một thư viện giao diện hiện đại cho Java Swing, giúp chương trình có giao diện đẹp hơn so với giao diện mặc định của Swing. Sẽ tích hợp FlatLaf và thêm biểu tượng cho các nút.

**Bước 1: Thêm FlatLaf vào pom.xml**

Cập nhật file pom.xml để thêm dependency cho FlatLaf:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>EmployeeManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

<dependencies>

*<!-- MySQL Connector -->*

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.33</version>

</dependency>

*<!-- FlatLaf -->*

<dependency>

<groupId>com.formdev</groupId>

<artifactId>flatlaf</artifactId>

<version>3.3</version>

</dependency>

</dependencies>

</project>

**Bước 2: Áp dụng FlatLaf trong Main.java**

Cập nhật Main.java để sử dụng FlatLaf làm giao diện.

package com.example;

import com.example.controller.EmployeeController;

import com.example.dao.DepartmentDAO;

import com.example.dao.EmployeeDAO;

import com.example.view.EmployeeView;

import com.formdev.flatlaf.FlatLightLaf;

import javax.swing.\*;

public class **Main** {

public static void main(String[] args) {

*// Áp dụng FlatLaf Look and Feel*

try {

UIManager.setLookAndFeel(new FlatLightLaf());

} catch (Exception e) {

e.printStackTrace();

}

*// Khởi tạo giao diện*

EmployeeView view = new EmployeeView();

EmployeeDAO employeeDAO = new EmployeeDAO();

DepartmentDAO departmentDAO = new DepartmentDAO();

new EmployeeController(view, employeeDAO, departmentDAO);

view.setVisible(true);

}

}

**Bước 3: Thêm biểu tượng cho các nút**

Để thêm biểu tượng cho các nút (ADD, UPDATE, DELETE, LOAD, CLEAR, SEARCH), bạn cần chuẩn bị các file ảnh biểu tượng (định dạng .png) và đặt chúng trong thư mục src/main/resources/icons/. Ví dụ:

* add.png
* update.png
* delete.png
* load.png
* clear.png
* search.png

**Lưu ý**: Có thể tải các biểu tượng miễn phí từ các trang như [Flaticon](https://www.flaticon.com/) hoặc [IconFinder](https://www.iconfinder.com/). Sau khi tải, đặt các file vào thư mục src/main/resources/icons/.

**Đã cập nhật EmployeeView.java**

Trong EmployeeView, đã thêm các biểu tượng cho các nút bằng cách sử dụng setIcon. Xem phần khởi tạo các nút trong buttonPanel và searchPanel.

**4. Tổng hợp các thay đổi**

**Thêm nút Clear**

* Thêm nút "CLEAR" vào EmployeeView trong buttonPanel.
* Xử lý sự kiện cho nút "CLEAR" trong EmployeeController, gọi phương thức clearFields() để xóa các trường nhập liệu và ô tìm kiếm.

**Tìm kiếm nâng cao (theo khoảng lương)**

* Cập nhật giao diện tìm kiếm trong EmployeeView để có hai ô Salary From và Salary To.
* Thêm phương thức searchBySalaryRange trong EmployeeDAO để tìm kiếm nhân viên trong khoảng lương.
* Xử lý sự kiện tìm kiếm theo khoảng lương trong EmployeeController, bao gồm kiểm tra đầu vào (giá trị phải là số dương, Salary From phải nhỏ hơn hoặc bằng Salary To).

**Cải thiện giao diện với FlatLaf**

* Thêm dependency FlatLaf vào pom.xml.
* Áp dụng FlatLaf Look and Feel trong Main.java.
* Thêm biểu tượng cho các nút trong EmployeeView (ADD, UPDATE, DELETE, LOAD, CLEAR, SEARCH).

**5. Chạy chương trình**

1. Đảm bảo bạn đã thiết lập cơ sở dữ liệu EMS như trong các phần trước.
2. Cập nhật thông tin USER và PASSWORD trong DatabaseConnection.java.
3. Tải và đặt các file biểu tượng (.png) vào thư mục src/main/resources/icons/.
4. Chạy file Main.java từ IntelliJ.

**6. Kết quả mong đợi**

* **Nút Clear**: Khi nhấn nút "**CLEAR**", tất cả các trường nhập liệu (**ID, Full Name, Gender, Birth Date, Salary**) và ô tìm kiếm (**Salary From, Salary To**) sẽ được xóa.
* **Tìm kiếm nâng cao**: Người dùng có thể tìm kiếm nhân viên theo khoảng lương (Salary From và Salary To). Chương trình sẽ hiển thị thông báo lỗi nếu:
  + Ô Salary From hoặc Salary To để trống.
  + Giá trị không phải là số hợp lệ.
  + Giá trị âm.
  + Salary From lớn hơn Salary To.
* **Giao diện đẹp hơn**:
  + Giao diện sẽ sử dụng FlatLaf, mang lại cảm giác hiện đại và mượt mà hơn so với giao diện mặc định của Swing.
  + Các nút (ADD, UPDATE, DELETE, LOAD, CLEAR, SEARCH) sẽ có biểu tượng tương ứng, làm tăng tính trực quan.

**7. Lưu ý**

* **Biểu tượng**: Nếu bạn không có các file biểu tượng, bạn có thể bỏ qua phần setIcon trong EmployeeView. Chương trình vẫn hoạt động bình thường, chỉ không có biểu tượng trên các nút.
* **FlatLaf**: Nếu bạn muốn thử các giao diện khác của FlatLaf (ví dụ: Dark Theme), bạn có thể thay FlatLightLaf trong Main.java bằng FlatDarkLaf hoặc các theme khác của FlatLaf.
* **Mở rộng thêm**: Nếu bạn muốn tiếp tục cải tiến, có thể:
  + Thêm tính năng xác nhận trước khi xóa (ví dụ: "Are you sure you want to delete this employee?").
  + Thêm tính năng export dữ liệu ra file Excel hoặc PDF.
  + Thêm phân quyền (admin và user) để hạn chế các tác vụ như xóa hoặc sửa.