Rajalakshmi Engineering College

Name: Alvin .B
Email: 240701034@rajalakshmi.edu.in
Roll no: 240701034
Phone: 9677000577
Branch: REC
Department: CSE - Section 10
Batch: 2028
Degree: B.E - CSE

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 11

Attempt : 1
Total Mark : 20
Marks Obtained : 20

Section 1 : Project

1.   Problem Statement

In ABC Corporation, employee records are stored in a database.

To efficiently manage employee details using Java and JDBC, you are tasked with building an Employee Management System that supports the following functionalities:

Adding a new employee

Updating an employee's salary

Viewing an employee's details

Displaying all employees

You are given two files:

File 1: Employee.java (POJO Class)

This class represents the Employee entity.

An Employee contains the following details:

Field	Description
employeeId	Unique Employee ID (Integer)
name	Employee Name (String)
department	Employee Department (String)
salary	Employee Salary (Double)

Students must write code in the marked area:

```java
class Employee {
    private int employeeId;
    private String name;
    private String department;
    private double salary;

    public Employee() {}

    public Employee(int employeeId, String name, String department, double salary) {
        // write your code here
    }

    // Include getters and setters
}
```

Expected in this part:

Assign parameter values to instance variables inside the constructor.

Add getters and setters for all attributes.

File 2: EmployeeDAO.java (Data Access Layer)

This class handles all database operations using JDBC.

Students must complete the missing JDBC logic in the following methods:

```java
class EmployeeDAO {

    public void addEmployee(Connection conn, Employee employee) throws
SQLException {

        // write your code here

    }

    public void updateSalary(Connection conn, int employeeId, double
newSalary) throws SQLException {

        // write your code here

    }

    public void deleteEmployee(Connection conn, int employeeId) throws
SQLException {

        // write your code here

    }

    public Employee viewEmployeeRecord(Connection conn, int employeeId)
throws SQLException {

        // write your code here

    }

    public List<Employee> displayAllEmployees(Connection conn) throws
SQLException {

        // write your code here

    }
```

```java
    private Employee mapToEmployee(ResultSet rs) throws SQLException {
        return new Employee(
            // write your code here
        );
    }
}
```

Expected in this part:

Write SQL queries for INSERT, UPDATE, DELETE, SELECT.

Execute queries using PreparedStatement or Statement.

Map ResultSet rows to Employee objects using mapToEmployee().

Return a List<Employee> where required.

The system should connect to a MySQL database using the following default credentials:

DB URL: jdbc:mysql://localhost/ri_dbUsername: testPassword: test123

The employees table has already been created with the following structure:

Input Format

The first line of input consists of an integer choice, representing the operation to be performed:

(1 for Add Employee, 2 for Update Salary, 3 for View Employee Record, 4 for Display All Employees, 5 for Exit)

For choice 1 (Add Employee):

1. The second line consists of an integer employee_id.
2. The third line consists of a string name.
3. The fourth line consists of a string department.
4. The fifth line consists of a double salary (must be at least 30000).

For choice 2 (Update Salary):

1. The second line consists of an integer employee_id.
2. The third line consists of a double new_salary (must be at least 30000).

For choice 3 (View Employee Record):

1. The second line consists of an integer employee_id.

For choice 4 (Display All Employees).

For choice 5 (Exit).

Output Format

For choice 1 (Add Employee),

1. Print "Employee added successfully" if the employee was added.

For choice 2 (Update Salary),

1. Print "Salary updated successfully" if the salary update was successful.
2. Print "Employee not found." if the specified employee ID does not exist.
3. Print "Salary must be at least 30000." if the provided salary is below the minimum.

For choice 3 (View Employee Record),

1. Display the employee details in the format:
2. ID: [employee_id] | Name: [name] | Department: [department] | Salary: [salary]
3. Print "Employee not found." if the specified employee ID does not exist.

For choice 4 (Display All Employees),

1. Display each employee on a new line in the format:
2. ID | Name | Department | Salary

For choice 5 (Exit),

1. Print "Exiting Employee Management System."

For invalid input:

1. Print "Invalid choice. Please try again."

Sample Test Case

Input: 1
101
Alice Johnson
Engineering
31000.75
4
6
5
Output: Employee added successfully
ID | Name | Department | Salary
101 | Alice Johnson | Engineering | 31000.75
Invalid choice. Please try again.
Exiting Employee Management System.

Answer

```java
import java.sql.*;
import java.util.Scanner;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

// Employee POJO Class
```

```java
class Employee {
    private int employeeId;
    private String name;
    private String department;
    private double salary;

    public Employee() {}

    public Employee(int employeeId, String name, String department, double
salary) {
        this.employeeId = employeeId;
        this.name = name;
        this.department = department;
        this.salary = salary;
    }

    // Getters and Setters
    public int getEmployeeId() {
        return employeeId;
    }

    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public double getSalary() {
```

```java
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}

// EmployeeDAO Class
class EmployeeDAO {

    public void addEmployee(Connection conn, Employee employee) throws
SQLException {
        String sql = "INSERT INTO employees (employee_id, name, department,
salary) VALUES (?, ?, ?, ?)";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, employee.getEmployeeId());
        pstmt.setString(2, employee.getName());
        pstmt.setString(3, employee.getDepartment());
        pstmt.setDouble(4, employee.getSalary());
        pstmt.executeUpdate();
        pstmt.close();
    }

    public void updateSalary(Connection conn, int employeeId, double newSalary)
throws SQLException {
        String sql = "UPDATE employees SET salary = ? WHERE employee_id = ?";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setDouble(1, newSalary);
        pstmt.setInt(2, employeeId);
        int rowsAffected = pstmt.executeUpdate();
        pstmt.close();

        if (rowsAffected == 0) {
            throw new SQLException("Employee not found");
        }
    }

    public void deleteEmployee(Connection conn, int employeeId) throws
SQLException {
        String sql = "DELETE FROM employees WHERE employee_id = ?";
        PreparedStatement pstmt = conn.prepareStatement(sql);
```

```java
        pstmt.setInt(1, employeeId);
        pstmt.executeUpdate();
        pstmt.close();
    }

    public Employee viewEmployeeRecord(Connection conn, int employeeId)
throws SQLException {
        String sql = "SELECT * FROM employees WHERE employee_id = ?";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, employeeId);
        ResultSet rs = pstmt.executeQuery();

        Employee employee = null;
        if (rs.next()) {
            employee = mapToEmployee(rs);
        }

        rs.close();
        pstmt.close();
        return employee;
    }

    public List<Employee> displayAllEmployees(Connection conn) throws
SQLException {
        String sql = "SELECT * FROM employees";
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql);

        List<Employee> employees = new ArrayList<>();
        while (rs.next()) {
            employees.add(mapToEmployee(rs));
        }

        rs.close();
        stmt.close();
        return employees;
    }

    private Employee mapToEmployee(ResultSet rs) throws SQLException {
        return new Employee(
            rs.getInt("employee_id"),
            rs.getString("name"),
```

```java
                rs.getString("department"),
                rs.getDouble("salary")
            );
        }
    }
}

// Main Class
class EmployeeManagementSystem {

    public static void addEmployee(Connection conn, Scanner scanner) {
        try {
            int employeeId = scanner.nextInt();
            scanner.nextLine(); // consume newline
            String name = scanner.nextLine();
            String department = scanner.nextLine();
            double salary = scanner.nextDouble();

            if (salary < 30000) {
                System.out.println("Salary must be at least 30000.");
                return;
            }

            Employee employee = new Employee(employeeId, name, department,
salary);
            EmployeeDAO dao = new EmployeeDAO();
            dao.addEmployee(conn, employee);
            System.out.println("Employee added successfully");
        } catch (SQLException e) {
            System.out.println("Database Error: " + e.getMessage());
        }
    }

    public static void updateSalary(Connection conn, Scanner scanner) {
        try {
            int employeeId = scanner.nextInt();
            double newSalary = scanner.nextDouble();

            if (newSalary < 30000) {
                System.out.println("Salary must be at least 30000.");
                return;
            }
```

```java
        EmployeeDAO dao = new EmployeeDAO();
        dao.updateSalary(conn, employeeId, newSalary);
        System.out.println("Salary updated successfully");
    } catch (SQLException e) {
        if (e.getMessage().equals("Employee not found")) {
            System.out.println("Employee not found.");
        } else {
            System.out.println("Database Error: " + e.getMessage());
        }
    }
}

public static void viewEmployeeRecord(Connection conn, Scanner scanner) {
    try {
        int employeeId = scanner.nextInt();

        EmployeeDAO dao = new EmployeeDAO();
        Employee employee = dao.viewEmployeeRecord(conn, employeeId);

        if (employee == null) {
            System.out.println("Employee not found.");
        } else {
            System.out.printf("ID: %d | Name: %s | Department: %s | Salary: %.2f%n",
                employee.getEmployeeId(),
                employee.getName(),
                employee.getDepartment(),
                employee.getSalary());
        }
    } catch (SQLException e) {
        System.out.println("Database Error: " + e.getMessage());
    }
}

public static void displayAllEmployees(Connection conn) {
    try {
        EmployeeDAO dao = new EmployeeDAO();
        List<Employee> employees = dao.displayAllEmployees(conn);

        System.out.println("ID | Name | Department | Salary");
        for (Employee emp : employees) {
            System.out.printf("%d | %s | %s | %.2f%n",
                emp.getEmployeeId(),
```

```java
                    emp.getName(),
                    emp.getDepartment(),
                    emp.getSalary());
            }
        } catch (SQLException e) {
            System.out.println("Database Error: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost/ri_db";
        String username = "test";
        String password = "test123";

        try (Connection conn = DriverManager.getConnection(url, username, password);
             Scanner scanner = new Scanner(System.in)) {

            int choice;
            do {
                choice = scanner.nextInt();

                switch (choice) {
                    case 1 -> addEmployee(conn, scanner);
                    case 2 -> updateSalary(conn, scanner);
                    case 3 -> viewEmployeeRecord(conn, scanner);
                    case 4 -> displayAllEmployees(conn);
                    case 5 -> System.out.println("Exiting Employee Management System.");
                    default -> System.out.println("Invalid choice. Please try again.");
                }

            } while (choice != 5);

        } catch (SQLException e) {
            System.out.println("Database Error: " + e.getMessage());
        }
    }
}
```

Status : Correct   *Marks : 10/10*

2.   Problem Statement

Create a JDBC-based Inventory Management System that handles runtime input to manage items in an inventory. The system should allow users to:

Add a new item (item ID, name, quantity, price).

Restock an item by increasing its quantity.

Reduce the stock of an item, ensuring sufficient quantity.

Display all items in the inventory in a sorted order by item ID.

Exit the application.

Half of the code is given here; Only the remaining part should be completed.

The system should connect to a MySQL database using the following default credentials:

DB URL: jdbc:mysql://localhost/ri_db

USER: test

PWD: test123

The items table has already been created with the following structure:

Table Name: items

Input Format

The first line of input consists of an integer choice, representing the operation to be performed (1 for Add Item, 2 for Restock item, 3 for reduce item, 4 for Display, 5 for Exit).

For choice 1 (Add Item):

- The second line consists of an integer item_id.

- The third line consists of a string name.
- The fourth line consists of an integer quantity.
- The fifth line consists of a double price.

For choice 2 (Restock Item):

- The second line consists of an integer item_id.
- The third line consists of an integer quantity_to_add (must be positive).

For choice 3 (Reduce Stock):

- The second line consists of an integer item_id.
- The third line consists of an integer quantity_to_remove (must be positive).

For choice 4 (Display Inventory):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

Output Format

For choice 1 (Add Item):

- Print "Item added successfully" if the item was added.
- Print "Failed to add item." if the insertion failed.

For choice 2 (Restock Item):

- Print "Item restocked successfully" if the restock was successful.
- Print "Item not found." if the specified item ID does not exist.

For choice 3 (Reduce Stock):

- Print "Stock reduced successfully" if the stock reduction was successful.
- Print "Not enough stock to remove." if there is insufficient quantity.
- Print "Item not found." if the specified item ID does not exist.

For choice 4 (Display Inventory):

- Display each item on a new line in the format:

- ID | Name | Quantity | Price
- If no items are available, print nothing (or handle with an appropriate message if desired).

For choice 5 (Exit):

- Print "Exiting Inventory Management System."

For invalid input:

- Print "Invalid choice. Please try again."

Sample Test Case

Input: 1
101
Laptop
50
1200.00
4
5
Output: Item added successfully
ID | Name | Quantity | Price
101 | Laptop | 50 | 1200.00
Exiting Inventory Management System.

Answer

```java
import java.sql.*;
import java.util.Scanner;

class InventoryManagementSystem {
    public static void main(String[] args) {
        try (Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/ri_db", "test", "test123");
             Scanner scanner = new Scanner(System.in)) {

            boolean running = true;

            while (running) {

                int choice = scanner.nextInt();

                switch (choice) {
```

```java
                case 1:
                    addItem(conn, scanner);
                    break;
                case 2:
                    restockItem(conn, scanner);
                    break;
                case 3:
                    reduceStock(conn, scanner);
                    break;
                case 4:
                    displayInventory(conn);
                    break;
                case 5:
                    System.out.println("Exiting Inventory Management System.");
                    running = false;
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}


public static void addItem(Connection conn, Scanner scanner) {
    try {
        int itemId = scanner.nextInt();
        scanner.nextLine(); // consume newline
        String name = scanner.nextLine();
        int quantity = scanner.nextInt();
        double price = scanner.nextDouble();

        String sql = "INSERT INTO items (item_id, name, quantity, price) VALUES
(?, ?, ?, ?)";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, itemId);
        pstmt.setString(2, name);
        pstmt.setInt(3, quantity);
        pstmt.setDouble(4, price);

        int rowsAffected = pstmt.executeUpdate();
```

```java
            if (rowsAffected > 0) {
                System.out.println("Item added successfully");
            } else {
                System.out.println("Failed to add item.");
            }
            pstmt.close();
        } catch (SQLException e) {
            System.out.println("Failed to add item.");
        }
    }

    public static void restockItem(Connection conn, Scanner scanner) {
        try {
            int itemId = scanner.nextInt();
            int quantityToAdd = scanner.nextInt();

            // First check if item exists
            String checkSql = "SELECT quantity FROM items WHERE item_id = ?";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            checkStmt.setInt(1, itemId);
            ResultSet rs = checkStmt.getResultSet();

            String updateSql = "UPDATE items SET quantity = quantity + ? WHERE
item_id = ?";
            PreparedStatement pstmt = conn.prepareStatement(updateSql);
            pstmt.setInt(1, quantityToAdd);
            pstmt.setInt(2, itemId);

            int rowsAffected = pstmt.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Item restocked successfully");
            } else {
                System.out.println("Item not found.");
            }

            pstmt.close();
            checkStmt.close();
        } catch (SQLException e) {
            System.out.println("Item not found.");
        }
    }
```

```java
public static void reduceStock(Connection conn, Scanner scanner) {
    try {
        int itemId = scanner.nextInt();
        int quantityToRemove = scanner.nextInt();

        // Check current quantity
        String checkSql = "SELECT quantity FROM items WHERE item_id = ?";
        PreparedStatement checkStmt = conn.prepareStatement(checkSql);
        checkStmt.setInt(1, itemId);
        ResultSet rs = checkStmt.executeQuery();

        if (rs.next()) {
            int currentQuantity = rs.getInt("quantity");
            if (currentQuantity >= quantityToRemove) {
                String updateSql = "UPDATE items SET quantity = quantity - ? WHERE item_id = ?";
                PreparedStatement pstmt = conn.prepareStatement(updateSql);
                pstmt.setInt(1, quantityToRemove);
                pstmt.setInt(2, itemId);
                pstmt.executeUpdate();
                System.out.println("Stock reduced successfully");
                pstmt.close();
            } else {
                System.out.println("Not enough stock to remove.");
            }
        } else {
            System.out.println("Item not found.");
        }

        rs.close();
        checkStmt.close();
    } catch (SQLException e) {
        System.out.println("Item not found.");
    }
}

public static void displayInventory(Connection conn) {
    try {
        String sql = "SELECT * FROM items ORDER BY item_id";
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql);
```

```java
        while (rs.next()) {
            int itemId = rs.getInt("item_id");
            String name = rs.getString("name");
            int quantity = rs.getInt("quantity");
            double price = rs.getDouble("price");

            System.out.printf("ID | Name | Quantity | Price%n");
            System.out.printf("%d | %s | %d | %.2f%n", itemId, name, quantity, price);
        }

        rs.close();
        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
  }
}
```

Status : Correct   **Marks : 10/10**