**Q1)**

The method calculateTax() is not a pure function because it relies on the instance variable rate, making the output dependent on the object's state. To make it pure, pass rate as a parameter so the method depends only on its inputs.

Way to make it pure:

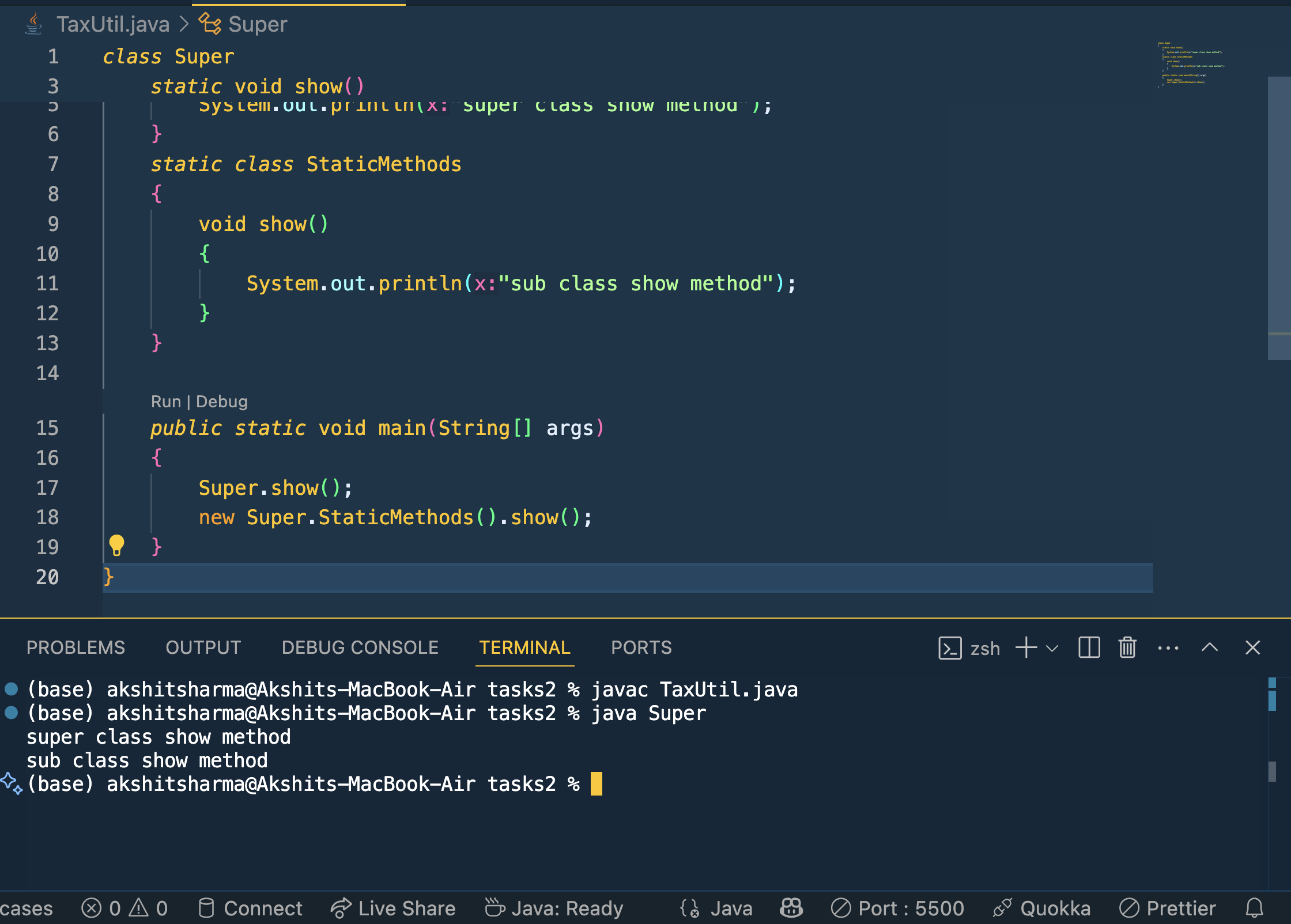
public class TaxUtil {

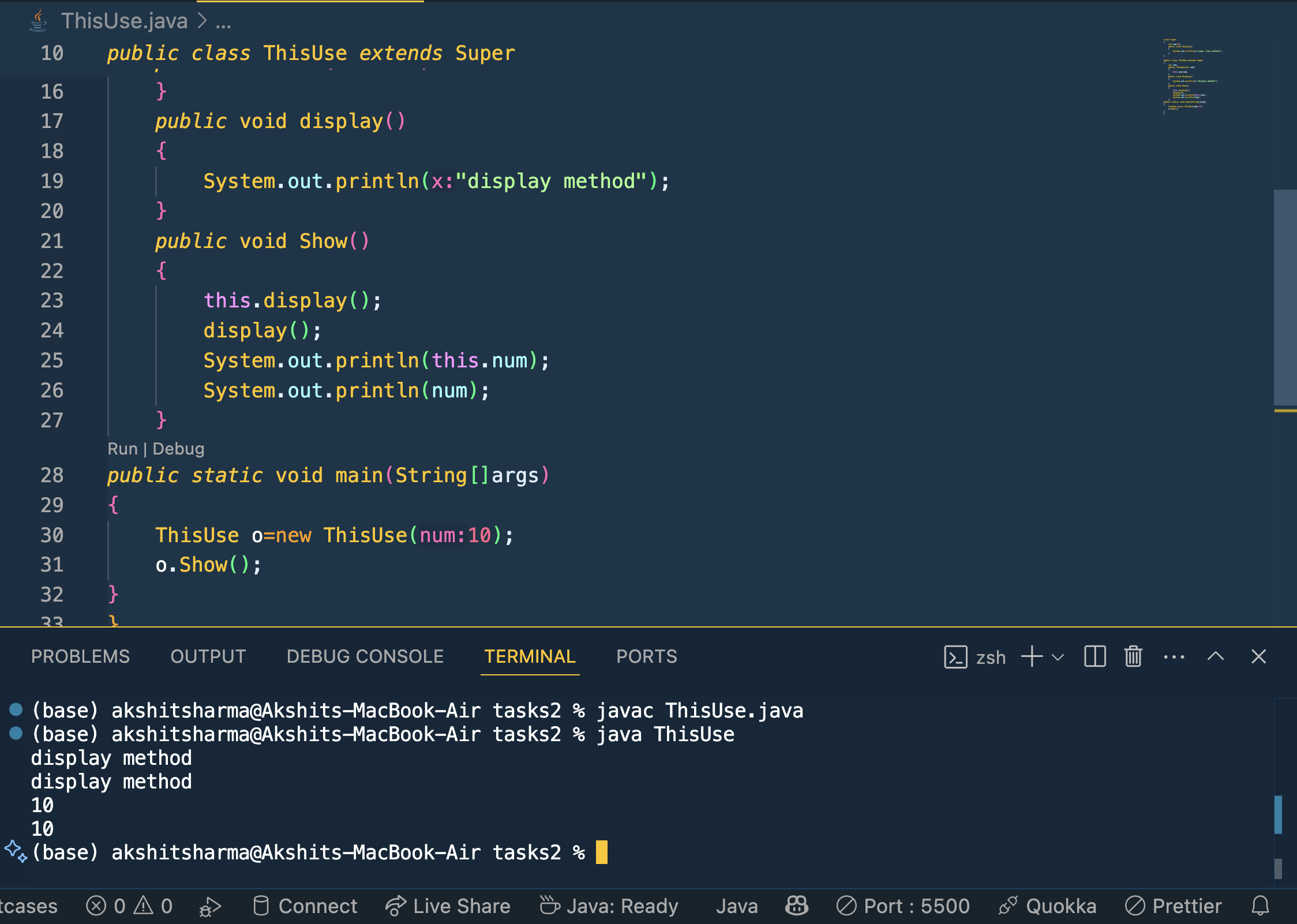
public static double calculateTax(double amount, double rate) {

return amount \* rate;

}

}

**Q2)**

**Q3)**

**Q4)**

The **Singleton design pattern** ensures that a class has **only one instance** throughout the application and provides a **global access point** to that instance. It is commonly used for managing shared resources like database connections or configuration managers.

public class Singleton {

// Step 1: Create a private static instance of the class

private static Singleton instance;

// Step 2: Make the constructor private so it can't be instantiated from outside

private Singleton() {}

// Step 3: Provide a public static method to access the instance

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton(); // lazy initialization

}

return instance;

}

public void showMessage() {

System.out.println("Hello from Singleton!");

}

}

**Q5)**

To ensure a class is **encapsulated**, we make its fields **private** and provide **public getter and setter methods** to control access and modification. This hides internal details and allows safe, controlled interaction with the class.

public class Person {

// Private fields (data hiding)

private String name;

private int age;

// Public getter and setter methods

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

if (age > 0) { // validation logic

this.age = age;

}

}

}

**Q6)**

public class Employee {

private int id;

private String name;

private String department;

public Employee(int id, String name, String department) {

this.id = id;

this.name = name;

this.department = department;

}

// Getters and setters

public int getId() { return id; }

public String getName() { return name; }

public String getDepartment() { return department; }

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

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

@Override

public String toString() {

return id + " - " + name + " - " + department;

}

}

import java.util.ArrayList;

public class EmployeeCRUD {

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

// Create

public void addEmployee(Employee e) {

employees.add(e);

}

// Read

public void viewEmployees() {

for (Employee e : employees) {

System.out.println(e);

}

}

// Update

public boolean updateEmployee(int id, String newName, String newDept) {

for (Employee e : employees) {

if (e.getId() == id) {

e.setName(newName);

e.setDepartment(newDept);

return true;

}

}

return false;

}

// Delete

public boolean deleteEmployee(int id) {

return employees.removeIf(e -> e.getId() == id);

}

// Main to test

public static void main(String[] args) {

EmployeeCRUD crud = new EmployeeCRUD();

crud.addEmployee(new Employee(1, "Alice", "HR"));

crud.addEmployee(new Employee(2, "Bob", "IT"));

System.out.println("All Employees:");

crud.viewEmployees();

crud.updateEmployee(2, "Bobby", "Finance");

System.out.println("\nAfter Update:");

crud.viewEmployees();

crud.deleteEmployee(1);

System.out.println("\nAfter Deletion:");

crud.viewEmployees();

}

}

**Q7)**

import java.sql.\*;

public class EmployeeJDBC {

private final String URL = "jdbc:mysql://localhost:3306/your\_db";

private final String USER = "root";

private final String PASS = "your\_password";

// Create

public void addEmployee(Employee e) throws SQLException {

String sql = "INSERT INTO employees (id, name, department) VALUES (?, ?, ?)";

try (Connection conn = DriverManager.getConnection(URL, USER, PASS);

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, e.getId());

stmt.setString(2, e.getName());

stmt.setString(3, e.getDepartment());

stmt.executeUpdate();

}

}

// Read

public void viewEmployees() throws SQLException {

String sql = "SELECT \* FROM employees";

try (Connection conn = DriverManager.getConnection(URL, USER, PASS);

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

System.out.println(rs.getInt("id") + " - " +

rs.getString("name") + " - " +

rs.getString("department"));

}

}

}

// Update

public void updateEmployee(int id, String name, String dept) throws SQLException {

String sql = "UPDATE employees SET name = ?, department = ? WHERE id = ?";

try (Connection conn = DriverManager.getConnection(URL, USER, PASS);

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, name);

stmt.setString(2, dept);

stmt.setInt(3, id);

stmt.executeUpdate();

}

}

// Delete

public void deleteEmployee(int id) throws SQLException {

String sql = "DELETE FROM employees WHERE id = ?";

try (Connection conn = DriverManager.getConnection(URL, USER, PASS);

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, id);

stmt.executeUpdate();

}

}

// Main to test

public static void main(String[] args) throws SQLException {

EmployeeJDBC db = new EmployeeJDBC();

db.addEmployee(new Employee(1, "Alice", "HR"));

db.addEmployee(new Employee(2, "Bob", "IT"));

System.out.println("All Employees:");

db.viewEmployees();

db.updateEmployee(2, "Bobby", "Finance");

System.out.println("\nAfter Update:");

db.viewEmployees();

db.deleteEmployee(1);

System.out.println("\nAfter Deletion:");

db.viewEmployees();

}

}

