PayXpert

Case Study 5

Name: Shardul Satish Kulkarni

SupersetId: 5270707

Database Schema:

1. Created Database schema with the given tables and their attributes
2. Also inserted mock records into the tables
3. Also added Foreign KEY and Primary KEY Constraints

```
-- Active: 1742545260664@@127.0.0.1@3306@payxpert
show databases;
Create Database PayXpert;
use PayXpert;
Create table employee(
    Employee_id int PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(255) NOT NULL,
    LastName VARCHAR(255) NOT NULL,
    DateOfBirth DATE NOT NULL,
    Gender ENUM('Male', 'Female') NOT NULL,
    Email VARCHAR(200) NOT NULL UNIQUE,
    PhoneNumber VARCHAR(20) NOT NULL UNIQUE,
    Address VARCHAR(100) NOT NULL,
    Position VARCHAR(100) NOT NULL,
    JoiningDate DATE NOT NULL,
    TerminationDate DATE DEFAULT NULL
    );
Create table Payroll(
    PayrollID INT PRIMARY KEY AUTO_INCREMENT,
    Employee_id INT NOT NULL,
    PayPeriodStartDate DATE NOT NULL,
    PayPeriodEndDate DATE NOT NULL,
    BasicSalary DECIMAL(10,2) NOT NULL,
    OvertimePay DECIMAL(10,2) NOT NULL,
```

```
    Deductions DECIMAL(10,2) NOT NULL,
    NetSalary DECIMAL(10,2) NOT NULL,
    FOREIGN KEY(Employee_id) REFERENCES Employee(Employee_id) ON DEL
);
Create table Tax(
    TaxID INT PRIMARY KEY AUTO_INCREMENT,
    Employee_id INT NOT NULL,
    TaxYear YEAR NOT NULL,
    TaxableIncome DECIMAL(10,2) NOT NULL,
    TaxAmount DECIMAL(10,2) NOT NULL,
    FOREIGN KEY(Employee_id) REFERENCES Employee(Employee_id) ON DEL
);

Create table FinancialRecord(
    RecordID INT PRIMARY KEY AUTO_INCREMENT,
    Employee_id INT NOT NULL,
    RecordDate DATE NOT NULL,
    Description VARCHAR(255) NOT NULL,
    Amount DECIMAL(10,2) NOT NULL,
    RecordType ENUM("income", "expense", "tax payment", "other") NOT NULL
    FOREIGN KEY(Employee_id) REFERENCES Employee(Employee_id) ON DEL
)



INSERT INTO Employee (FirstName, LastName, DateOfBirth, Gender, Email, Ph
('Ash', 'Mehta', '1992-04-15', 'Male', 'ash.mehta@gmail.com', '9876543210', 'M
('Shardul', 'Kulkarni', '1998-12-08', 'Male', 'shardul.kulkarni@gmail.com', '9123
('Virat', 'Kohli', '1988-11-05', 'Male', 'virat.kohli@gmail.com', '9000000001', 'De
('Neha', 'Rao', '1995-07-20', 'Female', 'neha.rao@gmail.com', '9000000002', '

INSERT INTO Payroll (Employee_id, PayPeriodStartDate, PayPeriodEndDate, B
(1, '2024-01-01', '2024-01-31', 50000.00, 2500.00, 5000.00, 47500.00),
(2, '2024-01-01', '2024-01-31', 45000.00, 2000.00, 4000.00, 43000.00),
(3, '2024-01-01', '2024-01-31', 80000.00, 5000.00, 10000.00, 75000.00),
(4, '2024-01-01', '2024-01-31', 40000.00, 1000.00, 3000.00, 38000.00);



INSERT INTO Tax (Employee_id, TaxYear, TaxableIncome, TaxAmount) VALUE
```

```
(1, 2024, 52500.00, 15750.00),
(2, 2024, 47000.00, 14100.00),
(3, 2024, 85000.00, 25500.00),
(4, 2024, 41000.00, 12300.00);



INSERT INTO FinancialRecord (Employee_id, RecordDate, Description, Amoun
(1, '2024-01-10', 'Performance Bonus', 3000.00, 'income'),
(1, '2024-01-20', 'Health Insurance', 2000.00, 'expense'),
(2, '2024-01-15', 'Salary Advance', 5000.00, 'expense'),
(3, '2024-01-18', 'Client Meeting Reimbursement', 2500.00, 'income'),
(4, '2024-01-25', 'TDS Payment', 12300.00, 'tax payment');
```

Created Directory Structure

1. Model: For the Entity Classes
2. Dao:
    a. Created Service Interfaces and Classes
    b. Created a DAO class in which technical implementation is present and in the service classes business logic will be present
3. Service: For Services
4. Exception: For Custom Exceptions
5. Util: For database Connection file
6. Tests: Contains Unit tests
7. Main: Contains main module

Created Entity Classes:

1. Created Entity classes with proper constructors and getter and setter functions.
2. In each class toString() method is overridden to print the correct details of entities instead of just class name and hash value.

1. Employee:

```
package payxpert.model;

import java.time.LocalDate;
import java.time.Period;
import java.time.ZoneId;
import java.util.Date;

public class Employee {
    private int EmployeeId;
    private String FirstName;
    private String LastName;
    private Date DateOfBirth;
    private String Gender;
    private String Email;
    private String PhoneNumber;
    private String Address;
    private String Position;
    private Date JoiningDate;
    private Date TerminationDate;

    public Employee(){}
    public Employee(String firstName, String lastName, Date dateOfBirth,
                    String gender, String email, String phoneNumber, String address,
                    String position, Date joiningDate, Date terminationDate) {
        this.FirstName = firstName;
        this.LastName = lastName;
        this.DateOfBirth = dateOfBirth;
        this.Gender = gender;
        this.Email = email;
        this.PhoneNumber = phoneNumber;
        this.Address = address;
        this.Position = position;
        this.JoiningDate = joiningDate;
        this.TerminationDate = terminationDate;
    }

    //Getter Functions
    public int getEmployeeId(){
```

```
    return EmployeeId;
}
public String getFirstName(){
    return FirstName;
}
public String getLastName(){
    return LastName;
}
public Date getDateOfBirth(){
    return DateOfBirth;
}
public String getGender(){
    return Gender;
}
public String getEmail(){
    return Email;
}
public String getPhoneNumber(){
    return PhoneNumber;
}

public String getAddress(){
    return Address;
}
public String getPosition(){
    return Position;
}
public Date getJoiningDate(){
    return JoiningDate;
}
public Date getTerminationDate(){
    return TerminationDate;
}
//Setter Functions

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

```
public void setFirstName(String FirstName) {
    this.FirstName = FirstName;
}

public void setLastName(String LastName){
    this.LastName = LastName;
}

public void setDateOfBirth(Date DateOfBirth){
    this.DateOfBirth = DateOfBirth;
}

public void setGender(String Gender){
    this.Gender = Gender;
}

public void setEmail(String Email){
    this.Email = Email;
}

public void setPhoneNumber(String PhoneNumber){
    this.PhoneNumber = PhoneNumber;
}

public void setAddress(String Address){
    this.Address = Address;
}

public void setPosition(String Position){
    this.Position = Position;
}

public void setJoiningDate(Date joiningDate) {
    this.JoiningDate = joiningDate;
}

public void setTerminationDate(Date terminationDate) {
```

```
        this.TerminationDate = terminationDate;
    }

    @Override
    public String toString() {
        return "Employee {" +
                "EmployeeId = " + EmployeeId +
                ", FirstName = '" + FirstName + '\'' +
                ", LastName = '" + LastName + '\'' +
                ", DateOfBirth = " + DateOfBirth +
                ", Gender = '" + Gender + '\'' +
                ", Email = '" + Email + '\'' +
                ", PhoneNumber = " + PhoneNumber +
                ", Address = '" + Address + '\'' +
                ", Position = '" + Position + '\'' +
                ", JoiningDate = " + JoiningDate +
                ", TerminationDate = " + TerminationDate +
                '}';
    }

    public int CalculateAge(){
        if(DateOfBirth == null){
            return 0;
        }

        LocalDate dob = DateOfBirth.toInstant().atZone(ZoneId.systemDefault()).
        LocalDate current = LocalDate.now();

        int age = Period.between(dob, current).getYears();

        return age;

    }

}
```

2. Payroll:

```
package payxpert.model;
import java.util.Date;

public class Payroll {
    private int payrollID;
    private int employeeID;
    private Date payPeriodStartDate;
    private Date payPeriodEndDate;
    private double basicSalary;
    private double overtimePay;
    private double deductions;
    private double netSalary;


    public Payroll() {
    }

    public Payroll(int payrollID, int employeeID, Date payPeriodStartDate, Date p
                double basicSalary, double overtimePay, double deductions, doubl
        this.payrollID = payrollID;
        this.employeeID = employeeID;
        this.payPeriodStartDate = payPeriodStartDate;
        this.payPeriodEndDate = payPeriodEndDate;
        this.basicSalary = basicSalary;
        this.overtimePay = overtimePay;
        this.deductions = deductions;
        this.netSalary = netSalary;
    }

    public int getPayrollID() {
        return payrollID;
    }

    public void setPayrollID(int payrollID) {
        this.payrollID = payrollID;
    }
```

```
public int getEmployeeID() {
    return employeeID;
}

public void setEmployeeID(int employeeID) {
    this.employeeID = employeeID;
}


public Date getPayPeriodStartDate() {
    return payPeriodStartDate;
}

public void setPayPeriodStartDate(Date payPeriodStartDate) {
    this.payPeriodStartDate = payPeriodStartDate;
}

public Date getPayPeriodEndDate() {
    return payPeriodEndDate;
}

public void setPayPeriodEndDate(Date payPeriodEndDate) {
    this.payPeriodEndDate = payPeriodEndDate;
}


public double getBasicSalary() {
    return basicSalary;
}

public void setBasicSalary(double basicSalary) {
    this.basicSalary = basicSalary;
}


public double getOvertimePay() {
    return overtimePay;
}
```

```
    public void setOvertimePay(double overtimePay) {
        this.overtimePay = overtimePay;
    }


    public double getDeductions() {
        return deductions;
    }

    public void setDeductions(double deductions) {
        this.deductions = deductions;
    }



    public double getNetSalary() {
        return netSalary;
    }

    public void setNetSalary(double netSalary) {
        this.netSalary = netSalary;
    }

    @Override
    public String toString(){
        return "Payroll {" +
                "PayrollId = " + payrollID +
                ", EmployeeId = " + employeeID +
                ", PayPeriodStartDate = " + payPeriodStartDate +
                ", PayPeriodEndDate = " + payPeriodEndDate +
                ", basicSalary = " + basicSalary +
                ", netSalary = " + netSalary +
                '}';
    }
}
```

3. FinancialRecord:

```
package payxpert.model;
import java.util.Date;

public class FinancialRecord {
    private int recordID;
    private int employeeID;
    private Date recordDate;
    private String description;
    private double amount;
    private String recordType;

    public FinancialRecord() {
    }

    public FinancialRecord(int recordID, int employeeID, Date recordDate, String
        this.recordID = recordID;
        this.employeeID = employeeID;
        this.recordDate = recordDate;
        this.description = description;
        this.amount = amount;
        this.recordType = recordType;
    }

    public int getRecordID() {
        return recordID;
    }

    public void setRecordID(int recordID) {
        this.recordID = recordID;
    }

    public int getEmployeeID() {
        return employeeID;
    }

    public void setEmployeeID(int employeeID) {
        this.employeeID = employeeID;
    }
```

```
public Date getRecordDate() {
    return recordDate;
}

public void setRecordDate(Date recordDate) {
    this.recordDate = recordDate;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public double getAmount() {
    return amount;
}

public void setAmount(double amount) {
    this.amount = amount;
}

public String getRecordType() {
    return recordType;
}

public void setRecordType(String recordType) {
    this.recordType = recordType;
}

@Override
public String toString(){
    return "FinancialRecord {" +
            "RecordId = " + recordID +
            ", EmployeeId = " + employeeID +
```

```
                ", RecordDate = " + recordDate +
                ", RecordType = " + recordType +
                ", Description = " + description +
                ", Amount = " + amount +
                '}';
    }

}
```

4. Tax:

```
package payxpert.model;

public class Tax {
    private int taxID;
    private int employeeID;
    private int taxYear;
    private double taxableIncome;
    private double taxAmount;


    public Tax() {
    }

    public Tax(int taxID, int employeeID, int taxYear, double taxableIncome, dou
        this.taxID = taxID;
        this.employeeID = employeeID;
        this.taxYear = taxYear;
        this.taxableIncome = taxableIncome;
        this.taxAmount = taxAmount;
    }


    public int getTaxID() {
        return taxID;
    }

    public void setTaxID(int taxID) {
```

```
    this.taxID = taxID;
}

public int getEmployeeID() {
    return employeeID;
}

public void setEmployeeID(int employeeID) {
    this.employeeID = employeeID;
}


public int getTaxYear() {
    return taxYear;
}

public void setTaxYear(int taxYear) {
    this.taxYear = taxYear;
}

public double getTaxableIncome() {
    return taxableIncome;
}

public void setTaxableIncome(double taxableIncome) {
    this.taxableIncome = taxableIncome;
}

public double getTaxAmount() {
    return taxAmount;
}

public void setTaxAmount(double taxAmount) {
    this.taxAmount = taxAmount;
}

@Override
public String toString(){
```

```
        return "Tax {" +
                "TaxID = " + taxID +
                ", EmployeeId = " + employeeID +
                ", TaxYear = " + taxYear +
                ", Taxable Income = " + taxableIncome +
                ", TaxAmount = " + taxAmount +
                '}';
    }
}
```

Created Service Provider Interfaces

IEmployeeService

```
package payxpert.dao;
import payxpert.exception.EmployeeNotFoundException;
import payxpert.exception.InvalidInputException;
import payxpert.model.Employee;

import java.util.List;

public interface IEmployeeService {
    Employee GetEmployeeById(int employeeId) throws EmployeeNotFoundExc

    List<Employee> GetAllEmployees() throws EmployeeNotFoundException;
    void AddEmployee(Employee employeeData) throws EmployeeNotFoundExc
    void UpdateEmployee(Employee employeeData) throws EmployeeNotFound
    void RemoveEmployee(int employeeId) throws EmployeeNotFoundExceptio
}
```

IFinancialRecordService

```
package payxpert.dao;

import payxpert.exception.FinancialRecordException;
import payxpert.model.FinancialRecord;

import java.util.Date;
```

```
import java.util.List;

public interface IFinancialRecordService {

    void AddFinancialRecord(int EmployeeId, String description, double amount
    FinancialRecord GetFinancialRecordById(int recordId) throws FinancialReco
    List<FinancialRecord> GetFinancialRecordsForEmployees(int EmployeeId) t
    List<FinancialRecord> GetFinancialRecordsForDate(Date recordDate) throw

}
```

IPayrollService

```
package payxpert.dao;

import payxpert.exception.PayrollGenerationException;
import payxpert.model.Payroll;

import java.util.Date;
import java.util.List;

public interface IPayrollService {
    Payroll GeneratePayroll(int EmployeeId, Date startDate, Date endDate) throv
    Payroll GetPayrollById(int payrollId) throws PayrollGenerationException;
    List<Payroll> GetPayrollsForEmployee(int EmployeeId) throws PayrollGener
    List<Payroll> GetPayrollsForPeriod(Date startDate, Date endDate) throws Pa


}
```

ITaxService

```
package payxpert.dao;

import payxpert.exception.TaxCalculationException;
import payxpert.model.Tax;

import java.util.List;
```

```
public interface ITaxService {
    double CalculateTax(int employeeId, int taxYear) throws TaxCalculationExce
    Tax GetTaxById(int taxId)throws TaxCalculationException;
    List<Tax> GetTaxesForEmployee(int employeeId) throws TaxCalculationExc
    List<Tax> GetTaxesForYear(int taxYear) throws TaxCalculationException;
}
```

Created Service classes & DAO that implement the above interface methods

1. Service classes are the ones that contain only the business logic like calculating tax, payroll that means business related things are present in this classes.
2. Technical implementation is done in the DAO classes for that they are called from these classes by using DAO Class's object.

EmployeeService

```
package payxpert.dao;

import payxpert.exception.EmployeeNotFoundException;
import payxpert.exception.InvalidInputException;
import payxpert.model.Employee;

import java.util.List;

public class EmployeeService implements IEmployeeService {
        private EmployeeDAO employeeDAO = new EmployeeDAO();

    @Override
    public Employee GetEmployeeById(int employeeId) throws EmployeeNotFou
        return employeeDAO.GetEmployeeById(employeeId);

    }
    @Override
    public List<Employee> GetAllEmployees() throws EmployeeNotFoundExcep
        return employeeDAO.GetAllEmployees();
```

```
    }
    @Override
    public void AddEmployee(Employee employeeData) throws EmployeeNotFo
        employeeDAO.AddEmployee(employeeData);

    }
    @Override
    public void RemoveEmployee(int EmployeeId) throws EmployeeNotFoundEx
        employeeDAO.RemoveEmployee(EmployeeId);

    }
    @Override
    public void UpdateEmployee(Employee employeeData) throws EmployeeNo
        employeeDAO.UpdateEmployee(employeeData);

    }
}
```

EmployeeDAO

```
package payxpert.dao;

import payxpert.exception.EmployeeNotFoundException;
import payxpert.exception.InvalidInputException;
import payxpert.model.Employee;
import payxpert.service.ValidationService;
import payxpert.util.DBConnection;

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

public class EmployeeDAO {

    private Connection con;

    public EmployeeDAO(){
```

```
    try{
        con = DBConnection.getConnection();
    }
    catch(SQLException e){
        e.printStackTrace();
    }
}

public Employee GetEmployeeById(int employeeId) throws EmployeeNotFou

    if(employeeId == 0){
        throw new EmployeeNotFoundException("Employee ID Can't be 0");
    }

    Employee emp = null;

    try{

        String query = "Select * from Employee WHERE EmployeeID = ?";
        PreparedStatement stmt = con.prepareStatement(query);
        stmt.setInt(1, employeeId);
        ResultSet rs = stmt.executeQuery();

        if(rs.next())
        {
            emp = new Employee();
            emp.setEmployeeId(rs.getInt("EmployeeId"));
            emp.setFirstName(rs.getString("FirstName"));
            emp.setLastName(rs.getString("LastName"));
            emp.setEmail(rs.getString("Email"));
            emp.setDateOfBirth(rs.getDate("DateOfBirth"));
            emp.setGender(rs.getString("Gender"));
            emp.setPhoneNumber(rs.getString("PhoneNumber"));
            emp.setAddress(rs.getString("Address"));
            emp.setPosition(rs.getString("Position"));
            emp.setJoiningDate(rs.getDate("JoiningDate"));
            emp.setTerminationDate(rs.getDate("TerminationDate"));
```

```
        }
        else{
            throw new EmployeeNotFoundException("Employee not found for Id
        }

        con.close();



    }
    catch (SQLException e){
        e.printStackTrace();
    }



    return emp;

}
public List<Employee> GetAllEmployees() throws EmployeeNotFoundExcep


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

    try{
        String sql = "Select * from Employee";
        PreparedStatement stmt = con.prepareStatement(sql);
        ResultSet rs = stmt.executeQuery();
        while(rs.next()){
            Employee emp = new Employee();

            emp.setEmployeeId(rs.getInt("EmployeeId"));
            emp.setFirstName(rs.getString("FirstName"));
            emp.setLastName(rs.getString("LastName"));
            emp.setEmail(rs.getString("Email"));
            emp.setDateOfBirth(rs.getDate("DateOfBirth"));
            emp.setGender(rs.getString("Gender"));
            emp.setPhoneNumber(rs.getString("PhoneNumber"));
```

```
            emp.setAddress(rs.getString("Address"));
            emp.setPosition(rs.getString("Position"));
            emp.setJoiningDate(rs.getDate("JoiningDate"));
            emp.setTerminationDate(rs.getDate("TerminationDate"));

            employees.add(emp);
        }
        if(employees.isEmpty()){
            throw new EmployeeNotFoundException("No Employee Records");
        }
        con.close();

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return employees;

}
public void AddEmployee(Employee EmployeeData) throws EmployeeNotFo
    if(ValidationService.isValidEmail(EmployeeData.getEmail())){
        throw new InvalidInputException("Invalid Email Format");
    }

    if(ValidationService.isValidPhoneNumber(EmployeeData.getPhoneNumbe
    {
        throw new InvalidInputException("Invalid Phone number");
    }

    if (ValidationService.isValidEmployee(EmployeeData)) {
        throw new InvalidInputException("Invalid Input NULL Data");
    }

    try{

        String sql = "Insert into Employee (FirstName, LastName, DateOfBirth,
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setString(1, EmployeeData.getFirstName());
        stmt.setString(2, EmployeeData.getLastName());
```

```
            stmt.setDate(3, new Date(EmployeeData.getDateOfBirth().getTime()));
            stmt.setString(4, EmployeeData.getGender());
            stmt.setString(5, EmployeeData.getEmail());
            stmt.setString(6, EmployeeData.getPhoneNumber());
            stmt.setString(7, EmployeeData.getAddress());
            stmt.setString(8, EmployeeData.getPosition());
            stmt.setDate(9, new Date(EmployeeData.getJoiningDate().getTime()));
            if (EmployeeData.getTerminationDate() != null) {
                stmt.setDate(10, new Date(EmployeeData.getTerminationDate().getT
            } else {
                stmt.setNull(10, java.sql.Types.DATE);
            }
            int rowsAdded = stmt.executeUpdate ();

            if(rowsAdded!=0){
                System.out.println("Added Successfully");
            }
            else{
                throw new EmployeeNotFoundException("Something went wrong wh
            }
            con.close();

        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }



    }
    public void UpdateEmployee(Employee employeeData) throws EmployeeNo
        if(ValidationService.isValidEmail(employeeData.getEmail())){
            throw new InvalidInputException("Invalid Email Format");
        }

        if(ValidationService.isValidPhoneNumber(employeeData.getPhoneNumbe
        {
```

```
        throw new InvalidInputException("Invalid Phone number");
    }

    if(ValidationService.isValidEmployee(employeeData)){
        throw new EmployeeNotFoundException("NULL DATA");
    }
    try{
        String sql = "UPDATE Employee SET FirstName = ?, LastName = ?, Dat
        PreparedStatement stmt = con.prepareStatement(sql);

        stmt.setString(1, employeeData.getFirstName());
        stmt.setString(2, employeeData.getLastName());
        stmt.setDate(3, new java.sql.Date(employeeData.getDateOfBirth().getT
        stmt.setString(4, employeeData.getGender());
        stmt.setString(5, employeeData.getEmail());
        stmt.setString(6, employeeData.getPhoneNumber());
        stmt.setString(7, employeeData.getAddress());
        stmt.setString(8, employeeData.getPosition());
        stmt.setDate(9, new java.sql.Date(employeeData.getJoiningDate().getT
        if (employeeData.getTerminationDate() != null) {
            stmt.setDate(10, new java.sql.Date(employeeData.getTerminationDat
        } else {
            stmt.setNull(10, java.sql.Types.DATE);
        }
        stmt.setInt(11, employeeData.getEmployeeId());

        int rows = stmt.executeUpdate();

        if(rows > 0){
            System.out.println("Employee Updated Successfully");
        }
        else{
            throw new EmployeeNotFoundException("Employee not found for id
        }
        con.close();

    }
    catch(SQLException e)
```

```
        {
            e.printStackTrace();
        }



    }
    public void RemoveEmployee(int EmployeeId) throws EmployeeNotFoundEx
        if(EmployeeId == 0 || EmployeeId < 0){
            throw new EmployeeNotFoundException("Employee ID should not be 0
        }
        try{
            String sql = "DELETE FROM Employee WHERE EmployeeID = ?";
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setInt(1, EmployeeId);
            int rowsAffected = stmt.executeUpdate();

            if(rowsAffected > 0){
                System.out.println("DELETED SUCCESSFULLY");
            }
            else{
                throw new EmployeeNotFoundException("Employee ID NOT FOUND
            }
            con.close();

        }
        catch(SQLException e){
            e.printStackTrace();
        }

    }

}
```

FinancialRecordService:

```
package payxpert.dao;
```

```
import payxpert.exception.FinancialRecordException;
import payxpert.model.FinancialRecord;

import java.util.Date;
import java.util.List;

public interface IFinancialRecordService {

    void AddFinancialRecord(int EmployeeId, String description, double amount
    FinancialRecord GetFinancialRecordById(int recordId) throws FinancialReco
    List<FinancialRecord> GetFinancialRecordsForEmployees(int EmployeeId) t
    List<FinancialRecord> GetFinancialRecordsForDate(Date recordDate) throw

}
```

FinancialRecordDAO

```
package payxpert.dao;

import payxpert.exception.FinancialRecordException;
import payxpert.model.FinancialRecord;
import payxpert.util.DBConnection;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class FinancialRecordDAO {
    private Connection con;
    public FinancialRecordDAO(){
        try{
            con = DBConnection.getConnection();
        }
```

```
        catch(SQLException e){
            e.printStackTrace();
        }
    }

    public void AddFinancialRecord(int EmployeeId, String description, double a
        // Edge Case: Check for null/empty description
        if (description == null || description.trim().isEmpty()) {
            throw new IllegalArgumentException("Description cannot be null or em
        }

        // Edge Case: Validate employee ID
        if (EmployeeId <= 0) {
            throw new IllegalArgumentException("Invalid employee ID");
        }

        // Edge Case: Validate amount
        if (amount <= 0) {
            throw new IllegalArgumentException("Amount must be greater than ze
        }

        // Edge Case: Validate recordType
        if (recordType == null || (!recordType.equalsIgnoreCase("income") && !re
            throw new IllegalArgumentException("Record type must be 'income' or
        }

        //Normalizing Record Types
        recordType = recordType.toLowerCase();

        try{


            String sql = "Insert into FinancialRecord(EmployeeId, RecordDate, Desc
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setInt(1, EmployeeId);
            stmt.setDate(2, java.sql.Date.valueOf(LocalDate.now()));
            stmt.setString(3, description);
            stmt.setDouble(4, amount);
```

```
            stmt.setString(5, recordType);
            int rowsAdded = stmt.executeUpdate();

            if(rowsAdded != 0){
                System.out.println("Record Added Successfully");
            }
            else{
                throw new FinancialRecordException("SOMETHING WENT WRONG \
            }
            con.close();

        }
        catch(SQLException e){
            e.printStackTrace();
            throw new FinancialRecordException("ERROR: "+ e.getMessage());

        }


    }

    public FinancialRecord GetFinancialRecordById(int recordId) throws Financi

        if(recordId == 0 || recordId < 0){
            throw new FinancialRecordException("Record Id Can't be 0 OR Negativ
        }
        FinancialRecord record = null;
        try{

            String sql = "Select * from FinancialRecord WHERE recordId = ?";
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setInt(1, recordId);
            ResultSet rs = stmt.executeQuery();
            while(rs.next()){
                record = new FinancialRecord();
                record.setRecordID(recordId);
                record.setRecordDate(rs.getDate("RecordDate"));
                record.setEmployeeID(rs.getInt("EmployeeId"));
```

```
            record.setDescription(rs.getString("Description"));
            record.setAmount(rs.getInt("Amount"));
            record.setRecordType(rs.getString("RecordType"));

        }
        if(record == null){
            throw new FinancialRecordException("NO RECORD FOUND FOR THE
        }

        con.close();

    }
    catch(SQLException e){
        e.printStackTrace();
        throw new FinancialRecordException("ERROR:" + e.getMessage());
    }
    return record;

}
public List<FinancialRecord> GetFinancialRecordsForEmployees(int Employ
    if(EmployeeId == 0 || EmployeeId < 0){
        throw new FinancialRecordException("EmployeeId can't be 0 OR Nega
    }

    FinancialRecord record = null;
    List<FinancialRecord> records = new ArrayList<>();
    try{

        String sql = "Select * from FinancialRecord WHERE EmployeeID = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, EmployeeId);
        ResultSet rs = stmt.executeQuery();
        while(rs.next()){
            record = new FinancialRecord();
            record.setRecordID(rs.getInt("RecordId"));
            record.setEmployeeID(EmployeeId);
            record.setRecordDate(rs.getDate("RecordDate"));
            record.setAmount(rs.getDouble("Amount"));
```

```
            record.setDescription(rs.getString("Description"));
            record.setRecordType(rs.getString("RecordType"));

            records.add(record);
        }
        if(records.isEmpty()){
            throw new FinancialRecordException("NO RECORD FOUND FOR THE
        }
        con.close();

    }
    catch(SQLException e){
        e.printStackTrace();
        throw new FinancialRecordException("ERROR:" + e.getMessage());
    }
    return records;

}
public List<FinancialRecord> GetFinancialRecordsForDate(Date recordDate)
    if(recordDate ==  null){
        throw new FinancialRecordException("NULL DATE");
    }

    FinancialRecord record = null;
    List<FinancialRecord> records = new ArrayList<>();

    try{

        String sql = "SELECT * FROM FinancialRecord WHERE RecordDate = ?"
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setDate(1, new java.sql.Date(recordDate.getTime()));
        ResultSet rs = stmt.executeQuery();
        while(rs.next()){
            record = new FinancialRecord();
            record.setRecordID(rs.getInt("RecordId"));
            record.setEmployeeID(rs.getInt("EmployeeId"));
            record.setRecordDate(recordDate);
            record.setDescription(rs.getString("Description"));
```

```
                record.setAmount(rs.getDouble("Amount"));
                record.setRecordType(rs.getString("RecordType"));

                records.add(record);
            }
            if(records.isEmpty()){
                throw new FinancialRecordException("NO Records found for the dat
            }
            con.close();


        }
        catch(SQLException e){
            e.printStackTrace();
            throw new FinancialRecordException("ERROR:" + e.getMessage());
        }
        return records;

    }
}
```

PayrollService

1. Changed method signature of the GeneratePayroll method, to calculate netSalary, we need OvertimePay
2. And for Overtimepay we need no of days the employee has worked overtime
3. So changed structure of the method, first we take input of basicSalary and no of days he has worked overtime.
4. Also the Overtimedays cannot be more than the days between start date and end date if they are then thrown payrollGenerationException
5. In service class I have written the business logic of calculating netSalary later I called the GeneratePayroll method in DAO class with all the parameters so that we can save the generated payroll in the database.

```
package payxpert.dao;
```

```
import payxpert.model.Payroll;

import java.time.LocalDate;
import java.time.ZoneId;
import java.time.temporal.ChronoUnit;
import java.util.Date;
import java.util.List;
import payxpert.exception.PayrollGenerationException;

public class PayrollService implements IPayrollService {
    PayrollDAO payrolldao = new PayrollDAO();
    @Override
    public Payroll GeneratePayroll(int employeeId, Date startDate, Date endDate
        //handling business logic of Calculating payroll
        //taken input of basic salary from the user as it is not given in the caseStu
        LocalDate localStart = startDate.toInstant().atZone(ZoneId.systemDefault
        LocalDate localEnd = endDate.toInstant().atZone(ZoneId.systemDefault()

        long workingDays = ChronoUnit.DAYS.between(localStart, localEnd) + 1;

        if(OverTimeDays > workingDays){
            throw new PayrollGenerationException("The working days should be w
        }

        double OvertimePay = OverTimeDays * 200; //taken input for how many d
        double deductions = basicSalary * 0.10;
        double netSalary = basicSalary + OvertimePay - deductions;

        return payrolldao.GeneratePayroll(employeeId, localStart, localEnd, basic

    }
    @Override
    public Payroll GetPayrollById(int payrollId) throws PayrollGenerationExceptio

        return payrolldao.GetPayrollById(payrollId);

    }
    @Override
```

```
    public List<Payroll> GetPayrollsForEmployee(int employeeId) throws Payrol

        return payrolldao.GetPayrollsForEmployee(employeeId);
    }
    @Override
    public List<Payroll> GetPayrollsForPeriod(Date startDate, Date endDate) thr
        return payrolldao.GetPayrollsForPeriod(startDate, endDate);
    }

}
```

PayrollDAO

```
package payxpert.dao;

import payxpert.exception.PayrollGenerationException;
import payxpert.model.Employee;
import payxpert.model.Payroll;
import payxpert.util.DBConnection;

import java.sql.*;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class PayrollDAO {
    private Connection con;
    public PayrollDAO(){
        try{
            con = DBConnection.getConnection();
        }
        catch(SQLException e){
            e.printStackTrace();
        }
    }


    public Payroll GeneratePayroll(int employeeId, LocalDate localStart, LocalDa
```

```
if (employeeId <= 0) {
    throw new PayrollGenerationException("Invalid employee ID.");
}
if (localStart == null || localEnd == null) {
    throw new PayrollGenerationException("Start and end dates cannot be
}
if (localEnd.isBefore(localStart)) {
    throw new PayrollGenerationException("End date cannot be before sta
}
if (basicSalary < 0 || OvertimePay < 0 || deductions < 0) {
    throw new PayrollGenerationException("Salary, overtime pay, or deduc
}
if (netSalary != (basicSalary + OvertimePay - deductions)) {
    throw new PayrollGenerationException("Net salary mismatch. Please v
}


Payroll payroll = null;
try{

    String sql = "INSERT INTO Payroll (Employee_Id, PayPeriodStartDate, P
    PreparedStatement stmt = con.prepareStatement(sql, Statement.RETU
    stmt.setInt(1, employeeId);
    stmt.setDate(2, Date.valueOf(localStart));
    stmt.setDate(3, Date.valueOf(localEnd));
    stmt.setDouble(4, basicSalary);
    stmt.setDouble(5, OvertimePay);
    stmt.setDouble(6, deductions);
    stmt.setDouble(7, netSalary);




    int rowsAdded = stmt.executeUpdate();
    ResultSet rs = stmt.getGeneratedKeys();
    if(rowsAdded > 0){
```

```
            if(rs.next()){
                int payrollId = rs.getInt(1);
                payroll = new Payroll();
                payroll.setPayrollID(payrollId);
            }
            System.out.println("Payroll Generated Successfully");
        }
        else{
            throw new PayrollGenerationException("SOMETHING WENT WRONG
        }
        payroll.setEmployeeID(employeeId);
        payroll.setPayPeriodStartDate(Date.valueOf(localStart));
        payroll.setPayPeriodEndDate(Date.valueOf(localEnd));
        payroll.setBasicSalary(basicSalary);
        payroll.setDeductions(deductions);
        payroll.setNetSalary(netSalary);
        con.close();

    }
    catch(SQLException e){
        e.printStackTrace();
    }

    return payroll;

}

public Payroll GetPayrollById(int payrollId) throws PayrollGenerationExcepti

    if(payrollId ==0 || payrollId < 0){
        throw new PayrollGenerationException("ID Can't be 0 OR Negative");
    }

    Payroll payroll = null;
    try{

        String sql = "Select * FROM Payroll WHERE PayrollID = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
```

```
            stmt.setInt(1, payrollId);

            ResultSet rs = stmt.executeQuery();
            payroll = new Payroll();
            if(rs.next()){
                payroll.setPayrollID(payrollId);
                payroll.setEmployeeID(rs.getInt("Employee_id"));
                payroll.setPayPeriodStartDate(rs.getDate("PayPeriodStartDate"));
                payroll.setPayPeriodEndDate(rs.getDate("PayPeriodEndDate"));
                payroll.setBasicSalary(rs.getDouble("BasicSalary"));
                payroll.setDeductions(rs.getDouble("Deductions"));
                payroll.setOvertimePay(rs.getDouble("OvertimePay"));
                payroll.setNetSalary(rs.getDouble("NetSalary"));

            }
            else{
                throw new PayrollGenerationException("Payroll NOT Found");
            }
            con.close();



        }
        catch (SQLException e){
            e.printStackTrace();
            throw new PayrollGenerationException("ERROR: " + e.getMessage());
        }
        return payroll;

    }

    public List<Payroll> GetPayrollsForEmployee(int employeeId) throws Payrol
        if(employeeId == 0 || employeeId < 0){
            throw new PayrollGenerationException("EmployeeId can't be 0 OR Neg
        }


        Payroll payroll = null;
```

```
    List<Payroll> payrolls = new ArrayList<>();
    try{

        String sql = "Select * from Payroll WHERE Employee_Id = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, employeeId);
        ResultSet rs = stmt.executeQuery();
        payroll = new Payroll();
        while(rs.next()){
            payroll.setPayrollID(rs.getInt("PayrollId"));
            payroll.setEmployeeID(employeeId);
            payroll.setBasicSalary(rs.getDouble("BasicSalary"));
            payroll.setPayPeriodStartDate(rs.getDate("PayPeriodStartDate"));
            payroll.setPayPeriodEndDate(rs.getDate("PayPeriodEndDate"));
            payroll.setDeductions(rs.getDouble("Deductions"));
            payroll.setOvertimePay(rs.getDouble("OvertimePay"));
            payroll.setNetSalary(rs.getDouble("Netsalary"));
            payrolls.add(payroll);
        }
        if(payrolls.isEmpty()){
            throw new PayrollGenerationException("Payroll Not Found for the En

        }
        con.close();


    }
    catch (SQLException e){
        e.printStackTrace();
        throw new PayrollGenerationException("ERROR: " + e.getMessage());
    }
    return payrolls;
}
public List<Payroll> GetPayrollsForPeriod(java.util.Date startDate, java.util.D

    if(startDate == null && endDate == null){
        throw new PayrollGenerationException("Dates can't be Null");
    }
```

```
    Payroll payroll = null;
    List<Payroll> payrolls = new ArrayList<>();

    try{

        String sql = "SELECT * FROM Payroll WHERE PayPeriodStartDate >= ?
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setDate(1, new Date(startDate.getTime()));
        stmt.setDate(2, new Date(endDate.getTime()));
        ResultSet rs = stmt.executeQuery();
        payroll = new Payroll();
        while(rs.next()){
            payroll.setPayrollID(rs.getInt("PayrollId"));
            payroll.setEmployeeID(rs.getInt("Employee_id"));
            payroll.setBasicSalary(rs.getDouble("BasicSalary"));
            payroll.setPayPeriodStartDate(rs.getDate("PayPeriodStartDate"));
            payroll.setPayPeriodEndDate(rs.getDate("PayPeriodEndDate"));
            payroll.setDeductions(rs.getDouble("Deductions"));
            payroll.setOvertimePay(rs.getDouble("OvertimePay"));
            payroll.setNetSalary(rs.getDouble("Netsalary"));
            payrolls.add(payroll);

        }
        if(payrolls.isEmpty()){
            throw new PayrollGenerationException("No records found for the giv
        }
        con.close();

    }
    catch(SQLException e){
        e.printStackTrace();
        throw new PayrollGenerationException("ERROR: " + e.getMessage());
    }
    return payrolls;

}
```

```
}
```

TaxService:

```
package payxpert.dao;

import payxpert.exception.TaxCalculationException;
import payxpert.model.Tax;

import java.util.List;

public class TaxService implements ITaxService {
    TaxDAO taxDAO = new TaxDAO();
    @Override
    public double CalculateTax(int employeeId, int taxYear) throws TaxCalculati
        return taxDAO.CalculateTax(employeeId, taxYear);
    }

    @Override
    public Tax GetTaxById(int taxId) throws TaxCalculationException{
        return taxDAO.GetTaxById(taxId);

    }

    @Override
    public List<Tax> GetTaxesForEmployee(int employeeId) throws TaxCalculat
        return taxDAO.GetTaxesForEmployee(employeeId);

    }

    @Override
    public List<Tax> GetTaxesForYear(int taxYear) throws TaxCalculationExcep
        return taxDAO.GetTaxesForYear(taxYear);

    }
}
```

TaxDAO:

```
package payxpert.dao;

import payxpert.exception.TaxCalculationException;
import payxpert.model.Tax;
import payxpert.util.DBConnection;

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 TaxDAO {
    private Connection con;
    public TaxDAO(){
        try{
            con = DBConnection.getConnection();
        }
        catch(SQLException e){
            e.printStackTrace();
        }
    }

    public double CalculateTax(int employeeId, int taxYear) throws TaxCalculati

        if(employeeId == 0 || employeeId < 0 && taxYear == 0 || taxYear < 0){
            throw new TaxCalculationException("VALUES CAN'T BE 0 OR NEGATIV
        }



        Tax tax = null;
        double taxAmount = 0;
        double taxableIncome = 0;
        try{
```

```
        String sql = "Select * from Payroll WHERE Employee_Id = ? AND YEAR(
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, employeeId);
        stmt.setInt(2, taxYear);
        ResultSet rs = stmt.executeQuery();
        while(rs.next()){
            double basicSalary = rs.getDouble("BasicSalary");
            double overtimePay = rs.getDouble("OvertimePay");
            double deductions = rs.getDouble("Deductions");

            taxableIncome += basicSalary + overtimePay - deductions; //all the

        }
        taxAmount = taxableIncome * 0.10;

        String mainsql = "INSERT INTO Tax (EmployeeId, taxYear, taxableIncon
        PreparedStatement stmt2 = con.prepareStatement(mainsql);
        stmt2.setInt(1, employeeId);
        stmt2.setInt(2, taxYear);
        stmt2.setDouble(3, taxableIncome);
        stmt2.setDouble(4, taxAmount);

        int rowsAdded = stmt2.executeUpdate();
        if(rowsAdded > 0){
            System.out.println("Calculated TAX Successfully, TaxAmount is: " +
        }
        else{
            throw new TaxCalculationException("SOMETHING WENT WRONG");
        }

        con.close();



    } catch (SQLException e) {
        e.printStackTrace();
        throw new TaxCalculationException("ERROR:"+e.getMessage());
```

```
    }
    return taxAmount;


}


public Tax GetTaxById(int taxId) throws TaxCalculationException{

    if(taxId == 0 || taxId < 0){
        throw new TaxCalculationException("Tax Id can't be 0 or negative");
    }

    Tax tax = null;
    try{

        String sql = "Select * from Tax WHERE TaxId = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, taxId);
        ResultSet rs = stmt.executeQuery();

        if(rs.next()){
            tax = new Tax();
            tax.setTaxID(taxId);
            tax.setTaxYear(rs.getInt("TaxYear"));
            tax.setEmployeeID(rs.getInt("EmployeeId"));
            tax.setTaxAmount(rs.getDouble("TaxAmount"));
            tax.setTaxableIncome(rs.getDouble("TaxableIncome"));

        }
        else{
            throw new TaxCalculationException("TaxId not found");
        }
        con.close();



    } catch (SQLException e) {
```

```
            e.printStackTrace();
            throw new TaxCalculationException("ERROR:"+e.getMessage());
        }
        return tax;

    }


    public List<Tax> GetTaxesForEmployee(int employeeId) throws TaxCalculat
        if(employeeId == 0 || employeeId < 0){
            throw new TaxCalculationException("ID Can't be 0 OR Null");
        }


        Tax tax = null;
        List<Tax> taxes = new ArrayList<>();
        try{

            String sql = "SELECT * FROM Tax WHERE EmployeeId = ?";
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setInt(1, employeeId);
            ResultSet rs = stmt.executeQuery();

            while(rs.next())
            {
                tax = new Tax();
                tax.setTaxID(rs.getInt("TaxId"));
                tax.setTaxYear(rs.getInt("TaxYear"));
                tax.setEmployeeID(rs.getInt("EmployeeId"));
                tax.setTaxAmount(rs.getDouble("TaxAmount"));
                tax.setTaxableIncome(rs.getDouble("TaxableIncome"));

                taxes.add(tax);
            }
            if(taxes.isEmpty()){
                throw new TaxCalculationException("Taxes for EmployeeID" + emplo
            }
            con.close();
```

```
    } catch (SQLException e) {
        e.printStackTrace();
        throw new TaxCalculationException("ERROR:"+e.getMessage());
    }
    return taxes;

}

public List<Tax> GetTaxesForYear(int taxYear) throws TaxCalculationExcep

    if(taxYear == 0 || taxYear < 0){
        throw new TaxCalculationException("Year can't be 0 Or null");
    }

    Tax tax = null;
    List<Tax> taxes = new ArrayList<>();
    try{

        String sql = "SELECT * FROM Tax WHERE TaxYear = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, taxYear);
        ResultSet rs = stmt.executeQuery();

        while(rs.next())
        {
            tax = new Tax();
            tax.setTaxID(rs.getInt("TaxId"));
            tax.setTaxYear(rs.getInt("TaxYear"));
            tax.setEmployeeID(rs.getInt("EmployeeId"));
            tax.setTaxAmount(rs.getDouble("TaxAmount"));
            tax.setTaxableIncome(rs.getDouble("TaxableIncome"));

            taxes.add(tax);
        }
```

```
            if(taxes.isEmpty()){
                throw new TaxCalculationException("Taxes for Year" + taxYear + "N
            }
            con.close();


        } catch (SQLException e) {
            e.printStackTrace();
            throw new TaxCalculationException("ERROR:"+e.getMessage());
        }
        return taxes;

    }
}
```

Database Connection:

1. Created Database Connection getConnection() method.
2. First created a db.properties file in which there are all the properties to connect to the database
3. Created PropertyUtil class which reads the db.properties file and returns a connection string
4. Created a DBConUtil class in which there is getConnection method which returns con object with the database connection

DBPropertyUtil

```
package payxpert.util;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class DBPropertyUtil {
    public static String getPropertyString() {

        Properties prop = new Properties();
```

```
        try {
            FileInputStream fs = new FileInputStream("db.properties");
            prop.load(fs);

            fs.close();

        } catch (IOException e) {
            e.printStackTrace();
        }

        String hostname = prop.getProperty("hostname");
        String dbname = prop.getProperty("dbname");
        String username = prop.getProperty("username");
        String password = prop.getProperty("password");
        String port = prop.getProperty("port");

        String connectionString = "jdbc:mysql://" + hostname + ":" + port + "/" +
                "?user=" + username + "&password=" + password;

        return connectionString;


    }
}
```

DBConnection

```
package payxpert.util;

import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class DBConnection {
```

```
    public static Connection getConnection() throws SQLException{

        return DriverManager.getConnection(DBPropertyUtil.getPropertyString())
    }
}
```

Created Custom Exceptions

1. Created custom exceptions and thrown them at proper time

```
package payxpert.exception;

public class DatabaseConnectionException extends Exception {
    public DatabaseConnectionException(String message){
        super(message);
    }
}
```

```
package payxpert.exception;

public class EmployeeNotFoundException extends Exception {
    public EmployeeNotFoundException(String message){
        super(message);
    }
}
```

```
package payxpert.exception;
public class FinancialRecordException extends Exception {
    public FinancialRecordException(String message){
        super(message);
    }
}
```

```
package payxpert.exception;

public class InvalidInputException extends Exception {
    public InvalidInputException(String message){
        super(message);
    }
}
```

```
package payxpert.exception;

public class PayrollGenerationException extends Exception {
    public PayrollGenerationException(String message){
        super(message);
    }
}
```

```
package payxpert.exception;

public class TaxCalculationException extends Exception{
    public TaxCalculationException(String message){
        super(message);
    }
}
```

Created Unit Tests

Following are the Unit tests for the Invalid Employee DataInput

```
package payxpert.test;

import org.junit.jupiter.api.Test;
import payxpert.dao.EmployeeService;
import payxpert.exception.EmployeeNotFoundException;
import payxpert.exception.InvalidInputException;
import payxpert.model.Employee;
```

```
import java.util.Date;

import static org.junit.jupiter.api.Assertions.*;
class InvalidEmployeeInputTest {

    EmployeeService emp = new EmployeeService();
    @Test
    public void InvalidEmployeeIdInput(){
        assertThrows(EmployeeNotFoundException.class, ()->{
            emp.GetEmployeeById(0);
        });
    }

    @Test
    public void testInvalidEmailFormat() {
        Employee empp = new Employee();
        empp.setFirstName("Test");
        empp.setLastName("User");
        empp.setEmail("invalid-email");

        empp.setPhoneNumber("1234567890"); // Required field
        empp.setGender("Male");
        empp.setAddress("Some Address");
        empp.setPosition("Tester");

        // Set valid dates
        empp.setDateOfBirth(new Date());
        empp.setJoiningDate(new Date());
        empp.setTerminationDate(null); // optional field

        assertThrows(InvalidInputException.class, () -> {
            emp.AddEmployee(empp);
        });
    }

    @Test
    public void testMissingRequiredFields() {
        Employee empp = new Employee(); // Missing all fields
```

```
        assertThrows(InvalidInputException.class, () -> {
            emp.AddEmployee(empp);
        });
    }

}
```

Following are the Unit tests for the wrong payroll and tax calculation

```
package payxpert.test;


import org.junit.jupiter.api.Test;
import payxpert.dao.PayrollService;
import payxpert.exception.EmployeeNotFoundException;
import payxpert.exception.InvalidInputException;
import payxpert.model.Payroll;
import payxpert.model.Tax;

import java.util.Arrays;
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

class PayrollTest {




    @Test
    public void calculateGrossSalary(){
        double basicSalary = 5000.00;
        double overTimePay = 200.00;
        double expected = 5200.00;
        Payroll payroll = new Payroll();
        payroll.setBasicSalary(basicSalary);
```

```
    payroll.setOvertimePay(overTimePay);
    double actual = payroll.getBasicSalary() + payroll.getOvertimePay();
    assertEquals(expected, actual, 0.01);
}

@Test
public void calculateNetSalaryAfterDeductions(){
    double basic = 5000.00;
    double overtime = 200.00;
    double deductions  = 1000.00;
    double expected = 4200.00;

    Payroll payroll = new Payroll();
    payroll.setBasicSalary(basic);
    payroll.setDeductions(deductions);
    payroll.setOvertimePay(overtime);
    payroll.setNetSalary(payroll.getBasicSalary() + payroll.getOvertimePay()

    assertEquals(expected, payroll.getNetSalary(), 0.01);
}

@Test
public void ProcessPayrollForMultipleEmployees(){
    Payroll payroll1 = new Payroll();
    payroll1.setBasicSalary(50000);
    payroll1.setOvertimePay(5000);
    payroll1.setDeductions(10000);

    payroll1.setNetSalary(50000 + 5000 - 10000); // Expected: 45000

    Payroll payroll2 = new Payroll();
    payroll2.setBasicSalary(60000);
    payroll2.setOvertimePay(6000);
    payroll2.setDeductions(11000);
    payroll2.setNetSalary(60000 + 6000 - 11000); // Expected: 55000

    Payroll payroll3 = new Payroll();
    payroll3.setBasicSalary(70000);
```

```
        payroll3.setOvertimePay(7000);
        payroll3.setDeductions(12000);
        payroll3.setNetSalary(70000 + 7000 - 12000); // Expected: 65000

        List<Payroll> payrolls = Arrays.asList(payroll1, payroll2, payroll3);

        assertEquals(3, payrolls.size());

        // Calculate total net salary for all payrolls
        double totalNetSalary = 0.0;
        for (Payroll p : payrolls) {
            totalNetSalary += p.getNetSalary();
        }


        // Expected total = 45000 + 55000 + 65000 = 165000
        double expectedTotal = 165000;


        assertEquals(expectedTotal, totalNetSalary, 0.01);
    }

    @Test
    public void VerifyTaxCalculationForHighIncomeEmployee(){
        Tax tax = new Tax();
        tax.setEmployeeID(5);
        tax.setTaxableIncome(120000.0);

        tax.setTaxAmount(tax.getTaxableIncome() * 0.3);

        double expectedTax = 36000.0;
        assertEquals(expectedTax, tax.getTaxAmount(), 0.01, "Tax calculation for

    }
}
```

Main Module

Following is the main module where we can perform all the above implemented operations:

```
package payxpert.main;

import payxpert.dao.EmployeeService;
import payxpert.dao.FinancialRecordService;
import payxpert.dao.PayrollService;
import payxpert.dao.TaxService;
import payxpert.exception.*;
import payxpert.model.Employee;
import payxpert.model.FinancialRecord;
import payxpert.model.Payroll;
import payxpert.model.Tax;

import java.time.LocalDate;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Scanner;

public class MainModule {

    private static Date ConvertDate(int year, int month, int day){
        LocalDate inputDate = LocalDate.of(year, month, day);
        Date date = Date.from(inputDate.atStartOfDay(ZoneId.systemDefault()).to
        return date;
    }

    public static void main(String[] args) throws InvalidInputException {
        while(true){
            System.out.println("--");
            System.out.println("WELCOME TO PayXpert");
            System.out.println("--");
            System.out.println("1. Add Employee");
            System.out.println("2. Update Employee");
            System.out.println("3. Remove Employee");
```

```
System.out.println("4. View All Employees");
System.out.println("5. View Specific Employee");
System.out.println("6. Generate Payroll For Employee");
System.out.println("7. Get specific Payroll");
System.out.println("8. Get Payrolls for Specific Employee");
System.out.println("9. Get Payrolls for Period");
System.out.println("10. Calculate Tax for Employee");
System.out.println("11. Get Tax By TaxID");
System.out.println("12. Get Tax for Specific Employee");
System.out.println("13. Get Tax by Tax Year");
System.out.println("14. Add new Financial Record");
System.out.println("15. Get Financial Record");
System.out.println("16. Get Financial Record for Specific Employee");
System.out.println("17. Get Financial Record for Specific Date");
System.out.println("0. Exit");
System.out.println("Enter your option:");
Scanner sc = new Scanner(System.in);
int option = sc.nextInt();
sc.nextLine();
switch(option){
   case 1→{

      try {
         System.out.print("Enter First Name: ");
         String firstName = sc.nextLine();

         System.out.print("Enter Last Name: ");
         String lastName = sc.nextLine();


         System.out.println("Enter Date of Birth:");
         System.out.print("Year: ");
         int birthYear = sc.nextInt();
         System.out.print("Month: ");
         int birthMonth = sc.nextInt();
         System.out.print("Day: ");
         int birthDay = sc.nextInt();
         Date dob = ConvertDate(birthYear, birthMonth, birthDay);
```

```
                sc.nextLine();

                System.out.print("Enter Gender (Male/Female): ");
                String gender = sc.nextLine();

                System.out.print("Enter Email: ");
                String email = sc.nextLine();

                System.out.print("Enter Phone Number: ");
                String phone = sc.next();
                sc.nextLine();

                System.out.print("Enter Address: ");
                String address = sc.nextLine();

                System.out.print("Enter Position: ");
                String position = sc.nextLine();

                System.out.println("Enter Joining Date:");
                System.out.print("Year: ");
                int joinYear = sc.nextInt();
                System.out.print("Month: ");
                int joinMonth = sc.nextInt();
                System.out.print("Day: ");
                int joinDay = sc.nextInt();
                Date joiningDate = ConvertDate(joinYear, joinMonth, joinDay);

                sc.nextLine();
                System.out.print("Is Termination Date available? (yes/no): ");
                String hasTermination = sc.nextLine();
                Date terminationDate = null;
                if (hasTermination.equalsIgnoreCase("yes")) {
                    System.out.println("Enter Termination Date:");
                    System.out.print("Year: ");
                    int termYear = sc.nextInt();
                    System.out.print("Month: ");
                    int termMonth = sc.nextInt();
                    System.out.print("Day: ");
```

```
                int termDay = sc.nextInt();
                terminationDate = ConvertDate(termYear, termMonth, termDa
            }
            Employee employeeData = new Employee(
                    firstName,
                    lastName,
                    dob,
                    gender,
                    email,
                    phone,
                    address,
                    position,
                    joiningDate,
                    terminationDate
            );
            EmployeeService emp = new EmployeeService();
            emp.AddEmployee(employeeData);

        } catch (EmployeeNotFoundException | InvalidInputException e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    }
    case 2->{

        try {
            System.out.print("Enter Employee ID to update: ");
            int empId = sc.nextInt();
            sc.nextLine();

            System.out.print("Enter First Name: ");
            String firstName = sc.nextLine();

            System.out.print("Enter Last Name: ");
            String lastName = sc.nextLine();

            System.out.println("Enter Date of Birth:");
            System.out.print("Year: ");
            int birthYear = sc.nextInt();
```

```
System.out.print("Month: ");
int birthMonth = sc.nextInt();
System.out.print("Day: ");
int birthDay = sc.nextInt();
Date dob = ConvertDate(birthYear, birthMonth, birthDay);
sc.nextLine();

System.out.print("Enter Gender (Male/Female): ");
String gender = sc.nextLine();

System.out.print("Enter Email: ");
String email = sc.nextLine();

System.out.print("Enter Phone Number (10 digits): ");
String phoneStr = sc.next();

System.out.print("Enter Address: ");
String address = sc.nextLine();

System.out.print("Enter Position: ");
String position = sc.nextLine();

System.out.println("Enter Joining Date:");
System.out.print("Year: ");
int joinYear = sc.nextInt();
System.out.print("Month: ");
int joinMonth = sc.nextInt();
System.out.print("Day: ");
int joinDay = sc.nextInt();
Date joiningDate = ConvertDate(joinYear, joinMonth, joinDay);
sc.nextLine();

System.out.print("Does the employee have a termination date?
String hasTermination = sc.nextLine();
Date terminationDate = null;
if (hasTermination.equalsIgnoreCase("yes")) {
    System.out.println("Enter Termination Date:");
    System.out.print("Year: ");
```

```
                int termYear = sc.nextInt();
                System.out.print("Month: ");
                int termMonth = sc.nextInt();
                System.out.print("Day: ");
                int termDay = sc.nextInt();
                terminationDate = ConvertDate(termYear, termMonth, termDa
            }

            Employee employeeData = new Employee(
                    firstName,
                    lastName,
                    dob,
                    gender,
                    email,
                    phoneStr,
                    address,
                    position,
                    joiningDate,
                    terminationDate
            );
            employeeData.setEmployeeId(empId);

            EmployeeService empService = new EmployeeService();
            empService.UpdateEmployee(employeeData);

        } catch (EmployeeNotFoundException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Something went wrong: " + e.getMessage()
            e.printStackTrace();
        }

        sc.close();
    }

    case 3->{
        System.out.println("Enter employeeId of employee you want to rer
        int empId = sc.nextInt();
```

```
            EmployeeService emp = new EmployeeService();
            try{
                emp.RemoveEmployee(empId);

            }
            catch(EmployeeNotFoundException e){
                 System.out.println(e.getMessage());
            }

        }

        case 4->{
            System.out.println("All Employees:");
            EmployeeService emp = new EmployeeService();

            try{
                Employee employee = new Employee();
                List<Employee> employees =  emp.GetAllEmployees();
                for(Employee e: employees)
                {
                    System.out.println(e);
                }



            }
            catch(EmployeeNotFoundException e){
                 System.out.println(e.getMessage());
            }
        }

        case 5->{
            Employee employee = null;
            System.out.println("Enter id of Employee You want to see: ");
            int empId = sc.nextInt();
            EmployeeService emp = new EmployeeService();
            try{
                employee = emp.GetEmployeeById(empId);
                System.out.println(employee);
```

```
            }
            catch(EmployeeNotFoundException e)
            {

                System.out.println(e.getMessage());
            }

        }

        case 6→{
            System.out.println("Generating Payroll");
            System.out.println("Enter EmployeeId: ");
            int empId = sc.nextInt();
            System.out.println("Enter Start Date Year: ");
            int year = sc.nextInt();
            System.out.println("Enter start Date Month: ");
            int month = sc.nextInt();
            System.out.println("Enter start Date Day: ");
            int day = sc.nextInt();
            System.out.println("Enter end Date Year: ");
            int yearr = sc.nextInt();
            System.out.println("Enter end Date Month: ");
            int monthh = sc.nextInt();
            System.out.println("Enter end Date Day: ");
            int dayy = sc.nextInt();
            Date startDate = ConvertDate(year, month, day);
            Date endDate = ConvertDate(yearr, monthh, dayy);
            System.out.println("Enter Basic Salary of Employee: ");
            int basicSalary = sc.nextInt();
            sc.nextLine();
            System.out.println("Enter For how many days employee has done
            long OverTimeDays = sc.nextLong();
            PayrollService pay = new PayrollService();
            try{
                Payroll payroll = null;
                payroll = pay.GeneratePayroll(empId, startDate, endDate, basicS
                System.out.println(payroll);
            }
```

```
            catch(PayrollGenerationException e)
            {
                System.out.println(e.getMessage());
            }

        }
        case 7->{
            System.out.println("Enter PayRoll ID: ");
            int payrollId = sc.nextInt();

            PayrollService pay = new PayrollService();
            try{
                Payroll payroll = null;
                payroll = pay.GetPayrollById(payrollId);
                System.out.println(payroll);
            }
            catch(PayrollGenerationException e){
                System.out.println(e.getMessage());

            }

        }
        case 8->{
            System.out.println("Enter id of employee for payroll: ");
            int empId = sc.nextInt();

            PayrollService pay = new PayrollService();
            try{
                List<Payroll> payrolls = new ArrayList<>();
                payrolls = pay.GetPayrollsForEmployee(empId);
                for(Payroll p: payrolls){
                    System.out.println(p);
                }
            }
            catch(PayrollGenerationException e){
                System.out.println(e.getMessage());

            }
```

```
}

case 9->{
   System.out.println("Enter Start Date Year: ");
   int year = sc.nextInt();
   System.out.println("Enter start Date Month: ");
   int month = sc.nextInt();
   System.out.println("Enter start Date Day: ");
   int day = sc.nextInt();
   System.out.println("Enter end Date Year: ");
   int yearr = sc.nextInt();
   System.out.println("Enter end Date Month: ");
   int monthh = sc.nextInt();
   System.out.println("Enter end Date Day: ");
   int dayy = sc.nextInt();
   Date startDate = ConvertDate(year, month, day);
   Date endDate = ConvertDate(yearr, monthh, dayy);

   PayrollService pay = new PayrollService();
   try{
      List<Payroll> payrolls = new ArrayList<>();
      payrolls = pay.GetPayrollsForPeriod(startDate, endDate);

      for(Payroll p: payrolls){
         System.out.println(p);
      }
   }
   catch(PayrollGenerationException e){
        System.out.println(e.getMessage());

   }

}

case 10->{
   System.out.println("Enter id of employee for the tax: ");
   int empId = sc.nextInt();
```

```
            System.out.println("Enter Year: ");
            int year = sc.nextInt();
            TaxService tax = new TaxService();
            try{
              double taxI = tax.CalculateTax(empId, year);

               System.out.println("Taxable Income: " + taxI);
            }
            catch(TaxCalculationException t){
               System.out.println(t.getMessage());
            }
          }
          case 11->{
            System.out.println("Enter taxId: ");
            int taxId = sc.nextInt();
            TaxService tax = new TaxService();
            try{
               Tax taxx = tax.GetTaxById(taxId);
               System.out.println(taxx);

            }
            catch(TaxCalculationException t){
               System.out.println(t.getMessage());
            }

          }
          case 12-> {
            System.out.println("Enter employeeID: ");
            int empId = sc.nextInt();
            TaxService tax = new TaxService();
            try{
              List<Tax> taxes = tax.GetTaxesForEmployee(empId);
              for(Tax t: taxes){
                 System.out.println(t);
              }

            }
            catch(TaxCalculationException t){
```

```
                System.out.println(t.getMessage());
            }
        }
        case 13 ->{
            System.out.println("Enter year");
            int year = sc.nextInt();
            TaxService tax = new TaxService();
            try{
              List<Tax> taxes = tax.GetTaxesForYear(year);

              for(Tax t: taxes){
                 System.out.println(t);
              }

            }
            catch(TaxCalculationException t){
               System.out.println(t.getMessage());
            }
        }
        case 14 ->{
            System.out.println("Add new financial Record");
            System.out.println("Enter employeeId: ");
            int empId = sc.nextInt();
            sc.nextLine();
            System.out.println("Enter Description: ");
            String desc = sc.nextLine();
            System.out.println("Enter Amount: ");
            double amount = sc.nextDouble();
            sc.nextLine();
            System.out.println("Enter Record Type: ");
            String recordType = sc.nextLine();

            FinancialRecordService fin = new FinancialRecordService();
            try{
               fin.AddFinancialRecord(empId, desc, amount, recordType);
            }
            catch(FinancialRecordException f)
            {
```

```
                System.out.println(f.getMessage());
            }

        }
        case 15 ->{
            System.out.println("Getting RecordID: ");
            int recordID = sc.nextInt();

            FinancialRecordService fin = new FinancialRecordService();
            try{
                System.out.println(fin.GetFinancialRecordById(recordID));
            }
            catch(FinancialRecordException f)
            {
                System.out.println(f.getMessage());
            }
        }
        case 16 ->{
            System.out.println("Enter EmployeeId: ");
            int empId = sc.nextInt();

            FinancialRecordService fin = new FinancialRecordService();
            try{
                List<FinancialRecord> records = fin.GetFinancialRecordsForEm
                for(FinancialRecord f: records)
                {
                    System.out.println(f);
                }
            }
            catch(FinancialRecordException f)
            {
                System.out.println(f.getMessage());
            }

        }
        case 17 ->{
            System.out.println("Enter Start Date Year: ");
            int year = sc.nextInt();
```

```
            System.out.println("Enter start Date Month: ");
            int month = sc.nextInt();
            System.out.println("Enter start Date Day: ");
            int day = sc.nextInt();

            Date recordDate = ConvertDate(year, month, day);

            FinancialRecordService fin = new FinancialRecordService();
            try{
              List<FinancialRecord> records =  fin.GetFinancialRecordsForDate
              for(FinancialRecord f: records)
              {
                 System.out.println(f);
              }

            }
            catch(FinancialRecordException f)
            {
               System.out.println(f.getMessage());
            }
        }
        case 0->{
            System.out.println("--");
            System.out.println("Thanks for Visiting!! GoodByeeee😊");
            System.out.println("--");
            sc.close();
            System.exit(0);
        }




        default -> {
            throw new InvalidInputException("Invalid Input");
        }


    }
```

```
        }


        //ith paryant

    }
}
```

END