PayXpert, The Payroll Management System

Instructions:

- Submitting assignments should be a single file or through git hub link shared with trainer and hexavarsity.
- Each assignment builds upon the previous one, and by the end, you will have a comprehensive application implemented in Java/C#/Python with a strong focus on SQL schema design, control flow statements, loops, arrays, collections, and database interaction.
- Follow object-oriented principles throughout the Java programming assignments. Use classes and objects to model real-world entities, encapsulate data and behavior, and ensure code reusability.
- Throw user defined exception from method and handle in the main method.
- The following Directory structure is to be followed in the application.
  - **entity/model**
    - Create entity classes in this package. All entity class should not have any business logic.
  - **dao**
    - Create Service Provider interface/abstract class to showcase functionalities.
    - Create the implementation class for the above interface/abstract class with db interaction.
  - **exception**
    - Create user defined exceptions in this package and handle exceptions whenever needed.
  - **util**
    - Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.
    - Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object.
  - **main**
    - Create a class MainModule and demonstrate the functionalities in a menu driven application.

Key Functionalities:

Employee Management:

- CRUD operations for employee data, including personal details, position, and employment history.

Payroll Processing:

- Automated calculation of employee salaries and deductions.
- Generation of pay stubs for each pay period.

Tax Calculation:

- Automatic computation of taxes based on employee income and deductions.

Financial Reporting:

- Generation of financial reports, including income statements and tax summaries.

Create following tables in SQL Schema with appropriate class and write the unit test case for the application.

SQL Tables:

1. Employee Table:

- EmployeeID (Primary Key): Unique identifier for each employee.
- FirstName: First name of the employee.
- LastName: Last name of the employee.
- DateOfBirth: Date of birth of the employee.
- Gender: Gender of the employee.
- Email: Email address of the employee.
- PhoneNumber: Phone number of the employee.
- Address: Residential address of the employee.
- Position: Job title or position of the employee.
- JoiningDate: Date when the employee joined the company.
- TerminationDate: Date when the employee left the company (nullable).

2. Payroll Table:

- PayrollID (Primary Key): Unique identifier for each payroll record.
- EmployeeID (Foreign Key): Foreign key referencing the Employee table.
- PayPeriodStartDate: Start date of the pay period.
- PayPeriodEndDate: End date of the pay period.
- BasicSalary: Base salary for the pay period.
- OvertimePay: Additional pay for overtime hours.
- Deductions: Total deductions for the pay period.
- NetSalary: Net salary after deductions.

3. Tax Table:

- TaxID (Primary Key): Unique identifier for each tax record.
- EmployeeID (Foreign Key): Foreign key referencing the Employee table.
- TaxYear: Year to which the tax information applies.
- TaxableIncome: Income subject to taxation.
- TaxAmount: Amount of tax to be paid.

4. FinancialRecord Table:

- RecordID (Primary Key): Unique identifier for each financial record.
- EmployeeID (Foreign Key): Foreign key referencing the Employee table.
- RecordDate: Date of the financial record.
- Description: Description or category of the financial record.
- Amount: Monetary amount of the record (income, expense, etc.).
- RecordType: Type of financial record (income, expense, tax payment, etc.).

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors (default and parametrized) and getters,setters)

Classes:

- Employee:
    - Properties: EmployeeID, FirstName, LastName, DateOfBirth, Gender, Email, PhoneNumber, Address, Position, JoiningDate, TerminationDate
    - Methods: CalculateAge()
- Payroll:
    - Properties: PayrollID, EmployeeID, PayPeriodStartDate, PayPeriodEndDate, BasicSalary, OvertimePay, Deductions, NetSalary
- Tax:
    - Properties: TaxID, EmployeeID, TaxYear, TaxableIncome, TaxAmount
- FinancialRecord:
    - Properties: RecordID, EmployeeID, RecordDate, Description, Amount, RecordType
- EmployeeService (implements IEmployeeService):
    - Methods: GetEmployeeById, GetAllEmployees, AddEmployee, UpdateEmployee, RemoveEmployee
- PayrollService (implements IPayrollService):
    - Methods: GeneratePayroll, GetPayrollById, GetPayrollsForEmployee, GetPayrollsForPeriod
- TaxService (implements ITaxService):
    - Methods: CalculateTax, GetTaxById, GetTaxesForEmployee, GetTaxesForYear
- FinancialRecordService (implements IFinancialRecordService):
    - Methods: AddFinancialRecord, GetFinancialRecordById, GetFinancialRecordsForEmployee, GetFinancialRecordsForDate
- DatabaseContext:
    - A class responsible for handling database connections and interactions.
- ValidationService:
    - A class for input validation and business rule enforcement.
- ReportGenerator:
    - A class for generating various reports based on payroll, tax, and financial record data.

Interfaces/Abstract class:

- IEmployeeService:
    - GetEmployeeById(employeeId)
    - GetAllEmployees()
    - AddEmployee(employeeData)
    - UpdateEmployee(employeeData)
    - RemoveEmployee(employeeId)
- IPayrollService:
    - GeneratePayroll(employeeId, startDate, endDate)
    - GetPayrollById(payrollId)
    - GetPayrollsForEmployee(employeeId)
    - GetPayrollsForPeriod(startDate, endDate)

- ITaxService:
    - CalculateTax(employeeId, taxYear)
    - GetTaxById(taxId)
    - GetTaxesForEmployee(employeeId)
    - GetTaxesForYear(taxYear)
- IFinancialRecordService:
    - AddFinancialRecord(employeeId, description, amount, recordType)
    - GetFinancialRecordById(recordId)
    - GetFinancialRecordsForEmployee(employeeId)
    - GetFinancialRecordsForDate(recordDate)

Connect your application to the SQL database:

- Create a connection string that includes the necessary information to connect to your SQL Server database. This includes the server name, database name, authentication credentials, and any other relevant settings.
- Use the SqlConnection class to establish a connection to the SQL Server database.
- Once the connection is open, you can use the SqlCommand class to execute SQL queries.

Custom Exceptions:

EmployeeNotFoundException:

- Thrown when attempting to access or perform operations on a non-existing employee.

PayrollGenerationException:

- Thrown when there is an issue with generating payroll for an employee.

TaxCalculationException:

- Thrown when there is an error in calculating taxes for an employee.

FinancialRecordException:

- Thrown when there is an issue with financial record management.

InvalidInputException:

- Thrown when input data doesn't meet the required criteria.

DatabaseConnectionException:

- Thrown when there is a problem establishing or maintaining a connection with the database.

Unit Testing:

Create NUnit test cases for car rental System are essential to ensure the correctness and reliability of your system. Below are some example questions to guide the creation of NUnit test cases for various components of the system:

Test Case: CalculateGrossSalaryForEmployee

- Objective: Verify that the system correctly calculates the gross salary for an employee.

Test Case: CalculateNetSalaryAfterDeductions

- Objective: Ensure that the system accurately calculates the net salary after deductions (taxes, insurance, etc.).

Test Case: VerifyTaxCalculationForHighIncomeEmployee

- Objective: Test the system's ability to calculate taxes for a high-income employee.

Test Case: ProcessPayrollForMultipleEmployees

- Objective: Test the end-to-end payroll processing for a batch of employees.

Test Case: VerifyErrorHandlingForInvalidEmployeeData

- Objective: Ensure the system handles invalid input data gracefully.