Name: Ayush Padhy

Case Study: PayXpert

TASK 1: CREATING CLASSES

1. Employee:

```python
from datetime import datetime

6 usages
class Employee:
    def __init__(self, employeeID, firstName, lastName, dateOfBirth, gender, email, phoneNum, address, position, joiningDate, terminationDate):
        self._employeeID = employeeID
        self._firstName = firstName
        self._lastName = lastName
        self._dateOfBirth = dateOfBirth
        self._gender = gender
        self._email = email
        self._phoneNum = phoneNum
        self._address = address
        self._position = position
        self._joiningDate = joiningDate
        self._terminationDate = terminationDate

    def calculateAge(self):
        current = datetime.now()
        curr_year = current.year

        dob = self._dateOfBirth
        dob_obj = datetime.strptime(dob, __format: "%Y-%m-%d")
        dob_year = dob_obj.year

        return curr_year-dob_year - ((current.month, current.day)<(dob_obj.month,dob_obj.day))
```

2. Payroll:

```python
from Employee import Employee

class Payroll(Employee):
    def __init__(self, payrollID, employeeID, payPeriodStartDate, payPeriodEndDate, basicSalary, overtimePay,
                 deductions, netSalary, firstName, lastName, dateOfBirth, gender, email, phoneNum, address, position,
                 joiningDate, terminationDate):
        super().__init__(employeeID, firstName, lastName, dateOfBirth, gender, email, phoneNum, address, position,
                         joiningDate, terminationDate)
        self._payrollID= payrollID
        self._employeeID = employeeID
        self._payPeriodStartDate = payPeriodStartDate
        self._payPeriodEndDate = payPeriodEndDate
        self._basicSalary = basicSalary
        self._overtimePay = overtimePay
        self._deductions = deductions
        self._netSalary = netSalary
```

3. Tax

```python
from Employee import Employee


class Tax(Employee):
    def __init__(self, employeeID, firstName, lastName, dateOfBirth, gender, email, phoneNum, address, position,
                 joiningDate, terminationDate, taxID, taxYear, taxableIncome, taxAmount):
        super().__init__(employeeID, firstName, lastName, dateOfBirth, gender, email, phoneNum, address, position,
                         joiningDate, terminationDate)
        self.taxID = taxID
        self.taxYear = taxYear
        self.taxableIncome = taxableIncome
        self.taxAmount = taxAmount
```

4. Financial Record

```python
from Employee import Employee


class FinancialRecord(Employee):
    def __init__(self, employeeID, firstName, lastName, dateOfBirth, gender, email, phoneNum, address, position,
                 joiningDate, terminationDate, recordID, recordDate, Description, amount, recordType):
        super().__init__(employeeID, firstName, lastName, dateOfBirth, gender, email, phoneNum, address, position,
                         joiningDate, terminationDate)
        self.recordID = recordID
        self.recordDate = recordDate
        self.Description = Description
        self.amount = amount
        self.recordType = recordType
```

TASK 2: SERVICE CLASSES

1. Employee Service:

```python
from abc import ABC, abstractmethod

2 usages
class IEmployeeService(ABC):

    @abstractmethod
    def GetEmployeeByID(self, empID):
        pass

    @abstractmethod
    def GetAllEmployees(self):
        pass

    @abstractmethod
    def AddEmployee(self):
        pass

    @abstractmethod
    def UpdateEmployee(self):
        pass

    @abstractmethod
    def RemoveEmployee(self):
        pass
```

2. Financial Record Service:

```python
from abc import ABC, abstractmethod


2 usages
class IFinancialRecordService(ABC):

    @abstractmethod
    def AddFinancialRecord(self):
        pass

    @abstractmethod
    def GetFinancialRecordById(self):
        pass

    @abstractmethod
    def GetFinancialRecordsForEmployee(self):
        pass

    @abstractmethod
    def GetFinancialRecordsForDate(self):
        pass
```

3. Payroll Service

```python
from abc import ABC, abstractmethod


2 usages
class IPayrollService(ABC):

    @abstractmethod
    def GeneratePayroll(self, employeeID, startDate, endDate):
        pass

    @abstractmethod
    def GetPayrollById(self, payrollID):
        pass

    @abstractmethod
    def GetPayrollsForEmployee(self):
        pass

    @abstractmethod
    def GetPayrollsForPeriod(self):
        pass
```

4. Tax Service

```
from abc import ABC, abstractmethod


2 usages
class ITaxService(ABC):

    @abstractmethod
    def CalculateTax(self):
        pass

    @abstractmethod
    def GetTaxById(self, taxID):
        pass

    @abstractmethod
    def GetTaxesForEmployee(self, employeeID):
        pass

    @abstractmethod
    def GetTaxesForYear(self, taxYear):
        pass
```

TASK 3: IMPLEMENTATION CLASSES

1. Employee Service

```
import ...


2 usages
class EmployeeServiceImpl(IEmployeeService):

    def __init__(self, dbUtil):
        self.dbUtil = dbUtil

    2 usages
    def GetEmployeeByID(self, empID: int):
        query = "Select * from employee where EmployeeID = %s"
        values = (empID,)
        result = self.dbUtil.fetchOne(query, values)
        return result

    1 usage
    def GetAllEmployees(self):
        query = "Select * from employee"
        result = self.dbUtil.fetchAll(query)
        return result
```

```
1 usage
def AddEmployee(self):
    employee = {
        'empID': self.generateUniqueEmployeeID(),
        'firstName': input("Enter your first name:"),
        'lastName': input("Enter your last name:"),
        'dob': input("Enter your DOB in (YYYY-MM-DD) format: "),
        'gender': input("Enter your gender(Male or Female): "),
        'email': input("Enter your EmailID: "),
        'phone': input("Enter your Phone Number: "),
        'address': input("Enter your Address: "),
        'position': input("Enter your position: "),
        'joiningDate': datetime.now().strftime("%Y-%m-%d")
    }
    if not (self.checkEmailID(employee['email'])):
        raise Exception("EmailID exists!!!")
    if not (self.checkPhoneNumber(employee['phone'])):
        raise Exception("Phone Number exists!!!")

    query = "insert into employee(EmployeeID, FirstName, LastName, DateOfBirth, Gender, Email, PhoneNumber, Address, Position, JoiningDate) values(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
    values = (employee['empID'], employee['firstName'], employee['lastName'], employee['dob'], employee['gender'],
              employee['email'], employee['phone'], employee['address'], employee['position'],
              employee['joiningDate'])
    self.dbUtil.executeQuery(query, values)
    print("Employee Added!!!")
```

```
1 usage
def UpdateEmployee(self):
    empID = int(input("Enter your Employee ID: "))

    if self.GetEmployeeByID(empID) is None:
        raise Exception("EmployeeNotFountException")

    while True:
        print("What do you want to update?")
        print("1. Email")
        print("2. Phone Number")
        print("3. Address")
        print("4. Position")
        print("5. Termination Date")
        print("6. Exit")
        ch = int(input("Enter your choice: "))
        if ch == 1:
            new_email = input("Enter your new emailID: ")
            if not (self.checkEmailID(new_email)):
                raise Exception("Email ID exists!!!")
            query = "update employee set Email=%s where EmployeeID = %s"
            value = (new_email, empID)
            return self.dbUtil.executeQuery(query, value)
        elif ch == 2:
            new_phone = input("Enter your new phone number: ")
            if not (self.checkEmailID(new_phone)):
                raise Exception("Phone Number already exists!!!")
            query = "update employee set PhoneNumber=%s where EmployeeID = %s"
            value = (new_phone, empID)
            return self.dbUtil.executeQuery(query, value)
        elif ch == 4:
            new_address = input("Enter your new address: ")
            query = "update employee set Address=%s where EmployeeID = %s"
            value = (new_address, empID)
            return self.dbUtil.executeQuery(query, value)
        elif ch == 5:
            new_position = input("Enter your new position: ")
            query = "update employee set Position=%s where EmployeeID = %s"
            value = (new_position, empID)
            return self.dbUtil.executeQuery(query, value)
        elif ch == 6:
            terminationDate = input("Enter your termination date: ")
            query = "update employee set TerminationDate=%s where EmployeeID = %s"
            value = (terminationDate, empID)
            return self.dbUtil.executeQuery(query, value)
        else:
            break
```

```
1 usage
def RemoveEmployee(self):
    empID = int(input("Enter Employee ID which you want to remove: "))
    query = "delete from employee where EmployeeID = %s"
    value = (empID,)
    self.dbUtil.executeQuery(query, value)
    print("Employee removed")

1 usage
def get_no_of_employees(self):
    query = "select count(*) from employee"
    result = self.dbUtil.fetchOne(query)
    return result[0]

1 usage
def generateUniqueEmployeeID(self):
    empID = int(self.get_no_of_employees()) + 1
    return empID

3 usages
def checkEmailID(self, email):
    query = "select Email from employee"
    result = self.dbUtil.fetchAll(query)
    if email not in (res[0] for res in result):
        return True
    else:
        return False

1 usage
def checkPhoneNumber(self, phone):
    query = "select PhoneNumber from employee"
    result = self.dbUtil.fetchAll(query)
    if phone not in (res[0] for res in result):
        return True
    else:
        return False
```

2. Financial Record Service

```
2 usages
class FinancialRecordServiceImpl(IFinancialRecordService):
    def __init__(self, dbUtil):
        self.dbUtil = dbUtil

    1 usage
    def AddFinancialRecord(self):

        record = {
            'RecordID': self.generateUniqueFRID(),
            'EmployeeID': input("Enter employeeID: "),
            'RecordDate': datetime.now().strftime("%Y-%m-%d"),
            'Description': input("Enter Description: "),
        }
        amount = float(input("Enter the amount"))
        recordType = input("Enter Record Type: Income or Expense: ")
        if recordType == "Income":
            amount = 1 * amount
        else:
            amount = -1 * amount
        query = "insert into financialrecord values(%s, %s, %s, %s, %s, %s)"
        values = (
            record['RecordID'], record['EmployeeID'], record['RecordDate'], record['Description'], amount, recordType)
        return self.dbUtil.executeQuery(query, values)

    1 usage
    def GetFinancialRecordById(self):
        recordID = int(input("Enter the recordID: "))
        query = "select * from financialrecord where RecordID = %s"
        values = (recordID,)
        result = self.dbUtil.fetchOne(query, values)
        return result

    1 usage
    def GetFinancialRecordsForEmployee(self):
        employeeID = int(input("Enter the employeeID: "))
        query = "select * from financialrecord where RecordID = %s"
        values = (employeeID,)
        result = self.dbUtil.fetchAll(query, values)
        return result
```

```
    1 usage
    def GetFinancialRecordsForDate(self):
        recordDate = (input("Enter the Record Date in (YYYY-MM-DD) format: "))
        query = "select * from financialrecord where RecordDate = %s"
        values = (recordDate,)
        result = self.dbUtil.fetchAll(query, values)
        return result

    1 usage
    def get_no_of_records(self):
        query = "select count(*) from financialrecord"
        result = self.dbUtil.fetchOne(query)
        return result[0]

    1 usage
    def generateUniqueFRID(self):
        empID = int(self.get_no_of_records()) + 1
        return empID
```

3. Payroll Service

```
from IPayrollService import IPayrollService


2 usages
class PayrollServiceImpl(IPayrollService):
    def __init__(self, dbUtil):
        self.dbUtil = dbUtil

    1 usage
    def GeneratePayroll(self, employeeID, startDate, endDate):
        if self.GetEmployeeByID(employeeID) is None:
            raise Exception("EmployeeNotFoundException")
        query = '''SELECT
                        e.EmployeeID,
                        e.FirstName,
                        e.LastName,
                        p.PayPeriodStartDate,
                        p.PayPeriodEndDate,
                        p.BasicSalary,
                        p.OvertimePay,
                        p.Deductions,
                        p.NetSalary
                    FROM
                        Employee e
                    INNER JOIN
                        Payroll p ON e.EmployeeID = p.EmployeeID
                    WHERE
                        e.EmployeeID = %s
                        AND p.PayPeriodStartDate >= %s
                        AND p.PayPeriodEndDate <= %s
                '''
        values = (employeeID, startDate, endDate)
        result = self.dbUtil.fetchAll(query, values)
        return result

    1 usage
    def GetPayrollById(self, payrollID):
        query = "SELECT * FROM payroll WHERE PayrollID=%s"
        values = (payrollID,)
        result = self.dbUtil.fetchOne(query, values)
        return result
```

```
1 usage
def GetPayrollById(self, payrollID):
    query = "SELECT * FROM payroll WHERE PayrollID=%s"
    values = (payrollID,)
    result = self.dbUtil.fetchOne(query, values)
    return result

1 usage
def GetPayrollsForEmployee(self):
    employeeID = int(input("Enter the employeeID: "))
    query = "select * from payroll p join employee e on p.EmployeeID = e.EmployeeID where p.EmployeeID = %s"
    values = (employeeID,)
    return self.dbUtil.fetchAll(query, values)

1 usage
def GetPayrollsForPeriod(self):
    startDate = input("Enter the start date: ")
    endDate = input("Enter the end date: ")
    query = "select * from payroll p where p.PayPeriodStartDate >= %s and p.PayPeriodEndDate <= %s"
    values = (startDate, endDate)
    return self.dbUtil.fetchAll(query, values)

1 usage
def GetEmployeeByID(self, empID: int):
    query = "Select * from employee where EmployeeID = %s"
    values = (empID,)
    result = self.dbUtil.fetchOne(query, values)
    return result[0]
```

4. Tax Service

```
from ITaxService import ITaxService


2 usages
class TaxServiceImpl(ITaxService):
    def __init__(self, dbUtil):
        self.dbUtil = dbUtil

    1 usage
    def CalculateTax(self):
        employeeID = int(input("Enter EmployeeID: "))
        taxYear = int(input("Enter Tax Year: "))

        if self.GetEmployeeByID(employeeID) is None:
            raise Exception("EmployeeNotFoundException")

        query = "select sum(TaxAmount) from tax where EmployeeID = %s and TaxYear = %s group by EmployeeID, TaxYear"
        values = (employeeID, taxYear)
        result = self.dbUtil.fetchAll(query, values)
        return result

    1 usage
    def GetTaxById(self, taxID):
        query = "select EmployeeID, TaxAmount from tax where TaxID = %s"
        values = (taxID,)
        result = self.dbUtil.fetchOne(query, values)
        return result
```

```
1 usage
def GetTaxesForEmployee(self, employeeID):
    query = "select EmployeeID, TaxAmount from tax where EmployeeID = %s"
    values = (employeeID,)
    result = self.dbUtil.fetchAll(query, values)
    return result

1 usage
def GetTaxesForYear(self, taxYear):
    query = "select TaxYear, TaxAmount from tax where TaxYear = %s"
    values = (taxYear,)
    result = self.dbUtil.fetchAll(query, values)
    return result

1 usage
def GetEmployeeByID(self, empID: int):
    query = "Select * from employee where EmployeeID = %s"
    values = (empID,)
    result = self.dbUtil.fetchOne(query, values)
    return result[0]
```

TASK4: PAYXPERT APP

```
import ...


1 usage
class PayXpertApp:
    def main(self):
        try:
            dbutil = DBUtil(host='localhost', user='root', password='SQL@bunny11', port='3306', database='payxpert')
        except Exception as e:
            raise Exception(f"Error Connecting Server: {e}")
        esl = EmployeeServiceImpl(dbutil)
        psl = PayrollServiceImpl(dbutil)
        frsl = FinancialRecordServiceImpl(dbutil)
        tsl = TaxServiceImpl(dbutil)

        while True:
            print("Welcome to PayXpert!!!")
            print("Which service do you want to use?")
            print("1. Employee Service")
            print("2. Payroll Service")
            print("3. Financial Record Service")
            print("4. Tax Service")
            print("5. Exit App")
            ch = int(input("Enter: "))
```

```
if ch == 1:
    while True:
        print("Welcome to the Employee Service")
        print("What do you want to do?")
        print("1. Get All Employee Information")
        print("2. Get Employee By ID")
        print("3. Register Employee in the database")
        print("4. Update Employee Information")
        print("5. Remove Employee from Database")
        print("6. Exit Employee Service")
        sc = int(input("Enter: "))
        if sc == 1:
            print(esl.GetAllEmployees())
        elif sc == 2:
            empID = int(input("Enter employeeID: "))
            print(esl.GetEmployeeByID(empID))
        elif sc == 3:
            esl.AddEmployee()
        elif sc == 4:
            esl.UpdateEmployee()
        elif sc == 5:
            esl.RemoveEmployee()
        else:
            break
elif ch == 2:
    while True:
        print("Welcome to the Payroll Service")
        print("What do you want to do?")
        print("1. Generate Payroll Information")
        print("2. Get Payroll By ID")
        print("3. Get Payrolls For Employee")
        print("4. Get Payrolls For Period")
        print("5. Exit Teacher Portal")
        tc = int(input("Enter: "))
        if tc == 1:
            employeeID = int(input("Enter the employeeID: "))
            startDate = input("Enter the start date: ")
            endDate = input("Enter the end date: ")
            print(psl.GeneratePayroll(employeeID, startDate, endDate))
        elif tc == 2:
            payrollID = int(input("Enter the payrollID: "))
            print(psl.GetPayrollById(payrollID))
        elif tc == 3:
            print(psl.GetPayrollsForEmployee())
        elif tc == 4:
            print(psl.GetPayrollsForPeriod())
        else:
            break
```

```
        elif ch == 3:
            while True:
                print("Welcome to the Financial Record Service")
                print("1. Add a Financial Record")
                print("2. Get Financial Record By ID")
                print("3. Get Financial Records For Employee")
                print("4. Get Financial Records For Date")
                print("5. Exit Service")
                tc = int(input("Enter: "))
                if tc == 1:
                    frsl.AddFinancialRecord()
                elif tc == 2:
                    print(frsl.GetFinancialRecordById())
                elif tc == 3:
                    print(frsl.GetFinancialRecordsForEmployee())
                elif tc == 4:
                    print(frsl.GetFinancialRecordsForDate())
                else:
                    break
        elif ch == 4:
            while True:
                print("Welcome to the Tax Service")
                print("What do you want to do?")
                print("1. Calculate Total Tax")
                print("2. Get Tax By Id")
                print("3. Get Taxes For Employee")
                print("4. Get Taxes For Year")
                print("5. Exit Payment Service")
                tc = int(input("Enter: "))
                if tc == 1:
                    print(tsl.CalculateTax())
                elif tc == 2:
                    taxID = int(input("Enter TaxID: "))
                    print(tsl.GetTaxById(taxID))
                elif tc == 3:
                    employeeID = int(input("Enter the employeeID: "))
                    print(tsl.GetTaxesForEmployee(employeeID))
                elif tc == 4:
                    taxYear = int(input("Enter the Tax Year: "))
                    print(tsl.GetTaxesForYear(taxYear))
                else:
                    break
        else:
            dbutil.closeConnection()
            print("Thank You for using the portal!!!")
            break
```

TASK 5: DATABASE CONNECTION

```python
from mysql.connector import connect


4 usages
class DBUtil:
    def __init__(self, host, user, password, port, database):
        self.connection = connect(
            host=host,
            user=user,
            password=password,
            port=port,
            database=database
        )
        self.cursor = self.connection.cursor()

    23 usages (23 dynamic)
    def executeQuery(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            self.connection.commit()
        except Exception as e:
            print(f"Query Execution Error! {e}")
            self.connection.rollback()

    13 usages (13 dynamic)
    def fetchAll(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            return self.cursor.fetchall()
        except Exception as e:
            print(f"Couldn't fetch all data! {e}")
            self.connection.rollback()

    def fetchOne(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            return self.cursor.fetchone()
        except Exception as e:
            print(f"Couldn't fetch one data! {e}")
            self.connection.rollback()

    1 usage
    def closeConnection(self):
        self.cursor.close()
        self.connection.close()
```

TASK 6: PYTHON TESTING

```python
class PayrollSystemTestCase(unittest.TestCase):

    def test_calculate_gross_salary_for_employee(self):
        expected_gross_salary = 5200
        actual_gross_salary = gross[0]
        self.assertEqual(actual_gross_salary, expected_gross_salary)

    def test_calculate_net_salary_after_deductions(self):
        expected_net_salary = 5000
        actual_net_salary = net[0]
        self.assertEqual(actual_net_salary, expected_net_salary)
```

```
========================================================================== 1 passed, 4 deselected in 0.12s ==========================================================================
PS F:\Companies\Hexaware\Technical Foundation Training\Python Training\PyCharm learning\PyCharm\CaseStudy\PayXpert\Classes> pytest UnitTesting.py -v -k gross
========================================================================== test session starts ==========================================================================
platform win32 -- Python 3.12.0, pytest-8.0.0, pluggy-1.4.0 -- C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
rootdir: F:\Companies\Hexaware\Technical Foundation Training\Python Training\PyCharm learning\PyCharm\CaseStudy\PayXpert\Classes
collected 5 items / 4 deselected / 1 selected

UnitTesting.py::PayrollSystemTestCase::test_calculate_gross_salary_for_employee PASSED                                                                                  [100%]

========================================================================== 1 passed, 4 deselected in 0.12s ==========================================================================
PS F:\Companies\Hexaware\Technical Foundation Training\Python Training\PyCharm learning\PyCharm\CaseStudy\PayXpert\Classes> pytest UnitTesting.py -v -k net
========================================================================== test session starts ==========================================================================
platform win32 -- Python 3.12.0, pytest-8.0.0, pluggy-1.4.0 -- C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
rootdir: F:\Companies\Hexaware\Technical Foundation Training\Python Training\PyCharm learning\PyCharm\CaseStudy\PayXpert\Classes
collected 5 items / 4 deselected / 1 selected

UnitTesting.py::PayrollSystemTestCase::test_calculate_net_salary_after_deductions PASSED                                                                               [100%]

========================================================================== 1 passed, 4 deselected in 0.11s ==========================================================================
```