PayXpert

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

```
from datetime import datetime
class Employee:
      def __init__(self, employee_id, first_name, last_name, date_of_birth, gender, email, phone_number, address, position, joining_date, termination_date):
        self.__employee_id = employee_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__date_of_birth = date_of_birth
        self.__gender = gender
        self.__email = email
        self.__phone_number = phone_number
        self.__address = address
        self.__position = position
        self.__joining_date = joining_date
        self.__termination_date = termination_date

    @property
    def e_id(self):
        return self.__employee_id
    @e_id.setter
    def e_id(self, value):
        self.__employee_id = value
```

```python
    @property
    def f_name(self):
        return self.__first_name

    @f_name.setter
    def f_name(self, value):
        self.__first_name = value

    @property
    def l_name(self):
        return self.__last_name

    @l_name.setter
    def l_name(self, value):
        self.__last_name = value

    @property
    def dob(self):
        return self.__date_of_birth

    @dob.setter
    def dob(self, value):
        self.__date_of_birth = value

    @property
    def gen(self):
        return self.__gender

    @gen.setter
    def gen(self, value):
        self.__gender = value

    @property
    def em(self):
        return self.__email

    @em.setter
    def em(self, value):
        self.__email = value

    @property
    def ph_no(self):
        return self.__phone_number

    @ph_no.setter
    def ph_no(self, value):
        self.__phone_number = value
```

```
    @property
    def add(self):
        return self.__address

    @add.setter
    def add(self, value):
        self.__address = value

    @property
    def pos(self):
        return self.__position

    @pos.setter
    def pos(self, value):
        self.__position = value

    @property
    def jd(self):
        return self.__joining_date

    @jd.setter
    def jd(self, value):
        self.__joining_date = value

    @property
    def td(self):
        return self.__termination_date

    @td.setter
    def td(self, value):
        self.__termination_date = value

    def calculate_age(self):
        today = datetime.today()
         age = today.year - self.__date_of_birth.year - ((today.month, today.day)
< (self.__date_of_birth.month, self.__date_of_birth.day))
        return age

class Payroll:
         def   __init__(self,   payroll_id,   employee_id,   pay_period_start_date,
pay_period_end_date, basic_salary, overtime_pay, deductions, net_salary):
        self.__payroll_id = payroll_id
        self.__employee_id = employee_id
        self.__pay_period_start_date = pay_period_start_date
        self.__pay_period_end_date = pay_period_end_date
        self.__basic_salary = basic_salary
        self.__overtime_pay = overtime_pay
        self.__deductions = deductions
        self.__net_salary = net_salary
```

```python
    @property
    def payroll_id(self):
        return self.__payroll_id

    @payroll_id.setter
    def payroll_id(self, value):
        self.__payroll_id = value

    @property
    def employee_id(self):
        return self.__employee_id

    @employee_id.setter
    def employee_id(self, value):
        self.__employee_id = value

    @property
    def pp_start_date(self):
        return self.__pay_period_start_date

    @pp_start_date.setter
    def pp_start_date(self, value):
        self.__pay_period_start_date = value

    @property
    def pp_end_date(self):
        return self.__pay_period_end_date

    @pp_end_date.setter
    def pp_end_date(self, value):
        self.__pay_period_end_date = value

    @property
    def basic_salary(self):
        return self.__basic_salary

    @basic_salary.setter
    def basic_salary(self, value):
        self.__basic_salary = value

    @property
    def overtime_pay(self):
        return self.__overtime_pay

    @overtime_pay.setter
    def overtime_pay(self, value):
        self.__overtime_pay = value
```

```
    @property
    def deductions(self):
        return self.__deductions

    @deductions.setter
    def deductions(self, value):
        self.__deductions = value

    @property
    def net_salary(self):
        return self.__net_salary

    @net_salary.setter
    def net_salary(self, value):
        self.__net_salary = value

class Tax:
        def __init__(self, tax_id, employee_id, tax_year, taxable_income,
tax_amount):
        self.__tax_id = tax_id
        self.__employee_id = employee_id
        self.__tax_year = tax_year
        self.__taxable_income = taxable_income
        self.__tax_amount = tax_amount

    @property
    def tax_id(self):
        return self.__tax_id

    @tax_id.setter
    def tax_id(self, value):
        self.__tax_id = value

    @property
    def employee_id(self):
        return self.__employee_id

    @employee_id.setter
    def employee_id(self, value):
        self.__employee_id = value

    @property
    def tax_year(self):
        return self.__tax_year

    @tax_year.setter
    def tax_year(self, value):
        self.__tax_year = value
```

```
    @property
    def taxable_income(self):
        return self.__taxable_income

    @taxable_income.setter
    def taxable_income(self, value):
        self.__taxable_income = value

    @property
    def tax_amount(self):
        return self.__tax_amount

    @tax_amount.setter
    def tax_amount(self, value):
        self.__tax_amount = value

class FinancialRecord:
    def __init__(self, record_id, employee_id, record_date, description, amount,
record_type):
        self.__record_id = record_id
        self.__employee_id = employee_id
        self.__record_date = record_date
        self.__description = description
        self.__amount = amount
        self.__record_type = record_type

    @property
    def record_id(self):
        return self.__record_id

    @record_id.setter
    def record_id(self, value):
        self.__record_id = value

    @property
    def employee_id(self):
        return self.__employee_id

    @employee_id.setter
    def employee_id(self, value):
        self.__employee_id = value

    @property
    def record_date(self):
        return self.__record_date

    @record_date.setter
    def record_date(self, value):
        self.__record_date = value
```

```python
    @property
    def description(self):
        return self.__description

    @description.setter
    def description(self, value):
        self.__description = value

    @property
    def amount(self):
        return self.__amount

    @amount.setter
    def amount(self, value):
        self.__amount = value

    @property
    def record_type(self):
        return self.__record_type

    @record_type.setter
    def record_type(self, value):
        self.__record_type = value

Employee1 = Employee(
    employee_id=101,
    first_name="Ilakiya",
    last_name="Rangaraju",
    date_of_birth=datetime(2002, 10, 27),
    gender="Female",
    email = "ilakiya@gmail.com",
    phone_number="9878898654",
    address="Kottaipudhur,Erode",
    position="Manager",
    joining_date=datetime(2023,1, 1),
    termination_date=None)
print("AGE IS : ",Employee1.calculate_age())
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/case study/case study 1.py"
AGE IS :  21

Process finished with exit code 0
```

EmployeeService (implements IEmployeeService):

- Methods:
  - GetEmployeeById
  - GetAllEmployees
  - AddEmployee
  - UpdateEmployee
  - RemoveEmployee

```
from at import IEmployeeService
from datetime import datetime
class EmployeeService(IEmployeeService):
   def __init__(self):
       self.employee_data = {
           1: {
               "employee_id": 101,
               "first_name": "Ilakiya",
               "last_name": "Rangaraju",
               "date_of_birth": datetime(2002, 10, 27),
               "gender": "Female",
               "email": "ilakiya@gmail.com",
               "phone_number": "9944049402",
               "address": "Erode",
               "position": "Manager",
               "joining_date": datetime(2020, 10, 21),
               "termination_date": None
           },
           2: {
               "employee_id": 102,
               "first_name": "Malar",
               "last_name": "Vizhi",
               "date_of_birth": datetime(1980, 5, 15),
               "gender": "Female",
               "email": "malar@gmail.com",
               "phone_number": "9790342951",
               "address": "Chennai",
               "position": "Analyst",
               "joining_date": datetime(2019, 6, 10),
               "termination_date": None
           },
           3: {
               "employee_id": 103,
               "first_name": "Ezhil",
               "last_name": "Rangaraju",
               "date_of_birth": datetime(1988, 8, 18),
```

```
            "gender": "Female",
            "email": "ezhil@gmail.com",
            "phone_number": "734526754",
            "address": "Bangalore",
            "position": "Engineer",
            "joining_date": datetime(2020, 3, 20),
            "termination_date": None
        },
        4: {
            "employee_id": 104,
            "first_name": "Jaswanth",
            "last_name": "Thangaraj",
            "date_of_birth": datetime(1989, 11, 5),
            "gender": "male",
            "email": "jaswanth.com",
            "phone_number": "9834215678",
            "address": "Pune",
            "position": "Manager",
            "joining_date": datetime(2016, 8, 23),
            "termination_date": None
        },
        5: {
            "employee_id": 105,
            "first_name": "Michael",
            "last_name": "Lee",
            "date_of_birth": datetime(1997, 3, 8),
            "gender": "Male",
            "email": "michael@example.com",
            "phone_number": "9995551234",
            "address": "Mumbai",
            "position": "Trainer",
            "joining_date": datetime(2010, 9, 1),
            "termination_date": None
        }
    }

def GetEmployeeById(self, employeeId):
    if employeeId in self.employee_data:
        return self.employee_data[employeeId]
    else:
        return None

def GetAllEmployees(self):
    return list(self.employee_data.values())

def AddEmployee(self, employeeData):
    employee_id = employeeData.get('employee_id')
    if employee_id:
        self.employee_data[employee_id] = employeeData
```

```python
            print("Employee added successfully")
        else:
            print("Failed to add employee")

    def UpdateEmployee(self, employeeData):
        employee_id = employeeData.get('employee_id')
        if employee_id and employee_id in self.employee_data:
            self.employee_data[employee_id] = employeeData
            print("Employee updated successfully")
        else:
            print("Failed to update employee")

    def RemoveEmployee(self, employeeId):
        if employeeId in self.employee_data:
            del self.employee_data[employeeId]
            print("Employee removed successfully")
        else:
            print("Failed to remove employee")


Employee_service1 = EmployeeService()

employee_data = {
    "employee_id": 101,
    "first_name": "Ilakiya",
    "last_name": "Rangaraju",
    "date_of_birth": datetime(2002, 10, 27),
    "gender": "Female",
    "email": "ilakiya@gmail.com",
    "phone_number": "9944049402",
    "address": "Erode",
    "position": "Manager",
    "joining_date": datetime(2020, 1, 1),
    "termination_date": None
}

print("---")

# 1
employee = Employee_service1.GetEmployeeById("1")
if employee:
    print("Employee details:")
    print("Employee ID:", employee["employee_id"])
    print("First Name:", employee["first_name"])
    print("Last Name:", employee["last_name"])
else:
    print("Employee not found")
# 2
Employee_service1.AddEmployee(employee_data)
```

```python
# 3
all_employees = Employee_service1.GetAllEmployees()
print("All Employees:")
for employee in all_employees:
    print("Employee ID:", employee["employee_id"])
    print("First Name:", employee["first_name"])
    print("Last Name:", employee["last_name"])

# 4
updated_employee_data = {
    "employee_id": 1,
    "first_name": "John",
    "last_name": "Smith",
    "date_of_birth": datetime(1990, 5, 20),
    "gender": "Male",
    "email": "john.smith@example.com",
    "phone_number": "1234567890",
    "address": "123 Main St, City",
    "position": "Manager",
    "joining_date": datetime(2020, 10, 21),
    "termination_date": None
}
Employee_service1.UpdateEmployee(updated_employee_data)

# 5
Employee_service1.RemoveEmployee(4)
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/case study/EmployeeService.py"
---
Employee not found
Employee added successfully
All Employees:
Employee ID: 101
First Name: Ilakiya
Last Name: Rangaraju
Employee ID: 102
First Name: Malar
Last Name: Vizhi
Employee ID: 103
First Name: Ezhil
Last Name: Rangaraju
Employee ID: 104
First Name: Jaswanth
Last Name: Thangaraj
Employee ID: 105
First Name: Michael
Last Name: Lee
Employee ID: 101
First Name: Ilakiya
Last Name: Rangaraju
Employee updated successfully
Employee removed successfully
```

PayrollService (implements IPayrollService):

- Methods:
  - GeneratePayroll
  - GetPayrollById
  - GetPayrollsForEmployee
  - GetPayrollsForPeriod

```python
from datetime import datetime
from at import IPayrollService
class PayrollService(IPayrollService):
   def __init__(self):
       self.payroll_data = {
           1: {
               "payroll_id": 1,
               "employee_id": 101,
               "pay_period_start_date": datetime(2024, 1, 1),
               "pay_period_end_date": datetime(2024, 1, 15),
               "basic_salary": 5000,
               "overtime_hours": 10,
               "overtime_rate": 20,
               "deductions": 1000,
               "net_salary": 4500
           },
           2: {
               "payroll_id": 2,
               "employee_id": 102,
               "pay_period_start_date": datetime(2024, 1, 16),
               "pay_period_end_date": datetime(2024, 1, 31),
               "basic_salary": 6000,
               "overtime_hours": 5,
               "overtime_rate": 25,
               "deductions": 1200,
               "net_salary": 5300
           },
           3: {
               "payroll_id": 3,
               "employee_id": 103,
               "pay_period_start_date": datetime(2024, 2, 1),
               "pay_period_end_date": datetime(2024, 2, 15),
               "basic_salary": 5500,
               "overtime_hours": 8,
               "overtime_rate": 20,
               "deductions": 800,
               "net_salary": 4700
           },
           4: {
```

```
        "payroll_id": 4,
        "employee_id": 104,
        "pay_period_start_date": datetime(2024, 2, 16),
        "pay_period_end_date": datetime(2024, 2, 29),
        "basic_salary": 5200,
        "overtime_hours": 12,
        "overtime_rate": 22,
        "deductions": 1100,
        "net_salary": 4800
    },
    5: {
        "payroll_id": 105,
        "employee_id": 5,
        "pay_period_start_date": datetime(2024, 3, 1),
        "pay_period_end_date": datetime(2024, 3, 15),
        "basic_salary": 5800,
        "overtime_hours": 7,
        "overtime_rate": 23,
        "deductions": 900,
        "net_salary": 5400
    },
    6: {
        "payroll_id": 6,
        "employee_id": 106,
        "pay_period_start_date": datetime(2024, 3, 16),
        "pay_period_end_date": datetime(2024, 3, 31),
        "basic_salary": 5300,
        "overtime_hours": 9,
        "overtime_rate": 21,
        "deductions": 1000,
        "net_salary": 4700
    },
    7: {
        "payroll_id": 7,
        "employee_id": 107,
        "pay_period_start_date": datetime(2024, 4, 1),
        "pay_period_end_date": datetime(2024, 4, 15),
        "basic_salary": 6100,
        "overtime_hours": 6,
        "overtime_rate": 24,
        "deductions": 1300,
        "net_salary": 5600
    },
    8: {
        "payroll_id": 8,
        "employee_id": 108,
        "pay_period_start_date": datetime(2024, 4, 16),
        "pay_period_end_date": datetime(2024, 4, 30),
        "basic_salary": 5400,
```

```
            "overtime_hours": 11,
            "overtime_rate": 21,
            "deductions": 950,
            "net_salary": 4900
        },
        9: {
            "payroll_id": 9,
            "employee_id": 109,
            "pay_period_start_date": datetime(2024, 5, 1),
            "pay_period_end_date": datetime(2024, 5, 15),
            "basic_salary": 5900,
            "overtime_hours": 8,
            "overtime_rate": 22,
            "deductions": 1000,
            "net_salary": 5300
        },
        10: {
            "payroll_id": 10,
            "employee_id": 110,
            "pay_period_start_date": datetime(2024, 5, 16),
            "pay_period_end_date": datetime(2024, 5, 31),
            "basic_salary": 5700,
            "overtime_hours": 10,
            "overtime_rate": 23,
            "deductions": 1100,
            "net_salary": 5200
        }
    }

def GeneratePayroll(self, employeeId, startDate, endDate):
    basic_salary = 50000
    overtime_hours = 10
    overtime_rate = 50
    deductions = 200
         net_salary = basic_salary + (overtime_hours * overtime_rate) -
deductions
    payroll_id = len(self.payroll_data) + 1
    payroll_details = {
        "payroll_id": payroll_id,
        "employee_id": employeeId,
        "pay_period_start_date": startDate,
        "pay_period_end_date": endDate,
        "basic_salary": basic_salary,
        "overtime_hours": overtime_hours,
        "overtime_rate": overtime_rate,
        "deductions": deductions,
        "net_salary": net_salary
    }
    self.payroll_data[payroll_id] = payroll_details
```

```
        return payroll_details

    def GetPayrollById(self, payrollId):
        return self.payroll_data.get(payrollId)

    def GetPayrollsForEmployee(self, employeeId):
        employee_payrolls = []
        for payroll_details in self.payroll_data.values():
            if payroll_details["employee_id"] == employeeId:
                employee_payrolls.append(payroll_details)
        return employee_payrolls

    def GetPayrollsForPeriod(self, startDate, endDate):
        payrolls_within_period = []
        for payroll_details in self.payroll_data.values():
            if startDate <= payroll_details["pay_period_start_date"] <= endDate:
                payrolls_within_period.append(payroll_details)
        return payrolls_within_period

    def CalculateGrossSalary(self, employee, basic_salary, overtime_pay):
        gross_salary = basic_salary + overtime_pay
        return gross_salary


Payroll_service1 = PayrollService()

# 1
employee_id = 101
start_date = datetime(2024, 4, 1)
end_date = datetime(2024, 4, 15)
generated_payroll  =  Payroll_service1.GeneratePayroll(employee_id,  start_date,
end_date)
print("Generated Payroll: ", generated_payroll)

# 2
payroll_id = 1
payroll_by_id = Payroll_service1.GetPayrollById(payroll_id)
print("\nPayroll by ID:", payroll_by_id)

# 3
payrolls_for_employee = Payroll_service1.GetPayrollsForEmployee(employee_id)
print("\nPayrolls for Employee:", payrolls_for_employee)

# 4
payrolls_within_period    =    Payroll_service1.GetPayrollsForPeriod(start_date,
end_date)
print("\nPayrolls within Period:", payrolls_within_period)
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/case study/PayrollService.py"
Generated Payroll:  {'payroll_id': 1, 'employee_id': 101, 'pay_period_start_date': datetime.datetime(2023, 4, 1, 0, 0), 'pay_period_end_date': datetime.datetim

Payroll by ID: {'payroll_id': 1, 'employee_id': 101, 'pay_period_start_date': datetime.datetime(2023, 4, 1, 0, 0), 'pay_period_end_date': datetime.datetime(202

Payrolls for Employee: [{'payroll_id': 1, 'employee_id': 101, 'pay_period_start_date': datetime.datetime(2023, 4, 1, 0, 0), 'pay_period_end_date': datetime.dat

Payrolls within Period: [{'payroll_id': 1, 'employee_id': 101, 'pay_period_start_date': datetime.datetime(2023, 4, 1, 0, 0), 'pay_period_end_date': datetime.da

Process finished with exit code 0
```

TaxService (implements ITaxService):

- Methods:
  - CalculateTax
  - GetTaxById
  - GetTaxesForEmployee

```python
from datetime import datetime
from at import ITaxService
class TaxService(ITaxService):
   def __init__(self):
       self.tax_data = {
            1: {"employee_id": 101, "tax_year": 2020, "taxable_income": 50000,
"tax_amount": 7500},
            2: {"employee_id": 102, "tax_year": 2020, "taxable_income": 60000,
"tax_amount": 9000},
            3: {"employee_id": 103, "tax_year": 2021, "taxable_income": 70000,
"tax_amount": 10500},
            4: {"employee_id": 104, "tax_year": 2021, "taxable_income": 55000,
"tax_amount": 8250},
            5: {"employee_id": 105, "tax_year": 2022, "taxable_income": 45000,
"tax_amount": 6750},
            6: {"employee_id": 106, "tax_year": 2022, "taxable_income": 80000,
"tax_amount": 12000},
            7: {"employee_id": 107, "tax_year": 2023, "taxable_income": 65000,
"tax_amount": 9750},
            8: {"employee_id": 108, "tax_year": 2023, "taxable_income": 75000,
"tax_amount": 11250},
            9: {"employee_id": 109, "tax_year": 2024, "taxable_income": 48000,
"tax_amount": 7200},
           10: {"employee_id": 110, "tax_year": 2024, "taxable_income": 90000,
"tax_amount": 13500}
       }
```

```python
    def CalculateTax(self, employeeId, taxYear):
        taxable_income = 60000
        tax_amount = 12000
        tax_id = len(self.tax_data) + 1
        tax_details = {
            "tax_id": tax_id,
            "employee_id": employeeId,
            "tax_year": taxYear,
            "taxable_income": taxable_income,
            "tax_amount": tax_amount
        }
        self.tax_data[tax_id] = tax_details
        return tax_details

    def GetTaxById(self, taxId):
        return self.tax_data.get(taxId)

    def GetTaxesForEmployee(self, employeeId):
        employee_taxes = []
        for tax_details in self.tax_data.values():
            if tax_details["employee_id"] == employeeId:
                employee_taxes.append(tax_details)
        return employee_taxes

    def GetTaxesForYear(self, taxYear):
        taxes_for_year = []
        for tax_details in self.tax_data.values():
            if tax_details["tax_year"] == taxYear:
                taxes_for_year.append(tax_details)
        return taxes_for_year


Tax_service1 = TaxService()
employee_id = 1
tax_year = 2024
calculated_tax = Tax_service1.CalculateTax(employee_id, tax_year)
print("\nCalculated Tax:", calculated_tax)
# 2
tax_id = 1
tax_by_id = Tax_service1.GetTaxById(tax_id)
print("\nTax by ID:", tax_by_id)

# 3
taxes_for_employee = Tax_service1.GetTaxesForEmployee(employee_id)
print("\nTaxes for Employee:", taxes_for_employee)

# 4
taxes_for_year = Tax_service1.GetTaxesForYear(tax_year)
print("\nTaxes for Year:", taxes_for_year)
```

```
TaxService
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/case study/TaxService.py"

Calculated Tax: {'tax_id': 1, 'employee_id': 1, 'tax_year': 2024, 'taxable_income': 60000, 'tax_amount': 12000}

Tax by ID: {'tax_id': 1, 'employee_id': 1, 'tax_year': 2024, 'taxable_income': 60000, 'tax_amount': 12000}

Taxes for Employee: [{'tax_id': 1, 'employee_id': 1, 'tax_year': 2024, 'taxable_income': 60000, 'tax_amount': 12000}]

Taxes for Year: [{'tax_id': 1, 'employee_id': 1, 'tax_year': 2024, 'taxable_income': 60000, 'tax_amount': 12000}]

Process finished with exit code 0
```

FinancialRecordService (implements IFinancialRecordService):

- Methods:
  - AddFinancialRecord
  - GetFinancialRecordById
  - GetFinancialRecordsForEmployee

```python
from datetime import datetime
from at import FinancialRecordService
class FinancialRecordService(FinancialRecordService):
   def __init__(self):
       self.financial_records = {
           1: {
               "record_id": 1,
               "employee_id": 101,
               "record_date": datetime(2024, 5, 1),
               "description": "Office supplies purchase",
               "amount": 320,
               "record_type": "Expense"
           },
           2: {
               "record_id": 2,
               "employee_id": 102,
               "record_date": datetime(2021, 11, 25),
               "description": "Internet bill",
               "amount": 148,
               "record_type": "Expense"
           },
           3: {
               "record_id": 3,
               "employee_id": 103,
               "record_date": datetime(2022, 12, 21),
               "description": "Salary payment",
               "amount": 4560,
```

```
    "record_type": "Income"
},
4: {
    "record_id": 4,
    "employee_id": 104,
    "record_date": datetime(2021, 7, 28),
    "description": "Travel expenses",
    "amount": 890,
    "record_type": "Expense"
},
5: {
    "record_id": 5,
    "employee_id": 105,
    "record_date": datetime(2022, 9, 23),
    "description": "Client meeting lunch",
    "amount": 950,
    "record_type": "Expense"
},
6: {
    "record_id": 6,
    "employee_id": 106,
    "record_date": datetime(2022, 8,12),
    "description": "Sales commission",
    "amount": 580,
    "record_type": "Income"
},
7: {
    "record_id": 7,
    "employee_id": 107,
    "record_date": datetime(2010, 7, 23),
    "description": "Office rent",
    "amount": 760,
    "record_type": "Expense"
},
8: {
    "record_id": 8,
    "employee_id": 108,
    "record_date": datetime(2022, 6, 28),
    "description": "Product sales",
    "amount": 2340,
    "record_type": "Income"
},
9: {
    "record_id": 9,
    "employee_id": 109,
    "record_date": datetime(2023, 3, 12),
    "description": "Training workshop fee",
    "amount": 876,
    "record_type": "Expense"
```

```
            },
            10: {
                "record_id": 10,
                "employee_id": 110,
                "record_date": datetime(2023, 9, 18),
                "description": "Consultation service income",
                "amount": 2340,
                "record_type": "Income"
            }
        }

    def AddFinancialRecord(self, employeeId, description, amount, recordType):
        record_id = len(self.financial_records) + 1
        record_date = datetime.now()
        financial_record = {
            "record_id": record_id,
            "employee_id": employeeId,
            "record_date": record_date,
            "description": description,
            "amount": amount,
            "record_type": recordType
        }
        self.financial_records[record_id] = financial_record
        return financial_record

    def GetFinancialRecordById(self, recordId):
        return self.financial_records.get(recordId)

    def GetFinancialRecordsForEmployee(self, employeeId):
        employee_records = []
        for record_id, record_details in self.financial_records.items():
            if record_details["employee_id"] == employeeId:
                employee_records.append(record_details)
        return employee_records

    def GetFinancialRecordsForDate(self, recordDate):
        records_for_date = []
        for record_details in self.financial_records.values():
            if record_details["record_date"].date() == recordDate:
                records_for_date.append(record_details)
        return records_for_date


FinancialRecordSservice1 = FinancialRecordService()
employee_id = 1
description = "Bonus"
amount = 10000
record_type = "Income"
# 1
```

```
added_record = FinancialRecordSservice1.AddFinancialRecord(employee_id,
description, amount, record_type)
print("\nAdded Financial Record:", added_record)
# 2
record_id = 1
record_by_id = FinancialRecordSservice1.GetFinancialRecordById(record_id)
print("\nFinancial Record by ID:", record_by_id)
# 3
records_for_employee =
FinancialRecordSservice1.GetFinancialRecordsForEmployee(employee_id)
print("\nFinancial Records for Employee:", records_for_employee)
# 4
record_date = datetime.now().date()
records_for_date =
FinancialRecordSservice1.GetFinancialRecordsForDate(record_date)
print("\nFinancial Records for Date:", records_for_date)
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/case study/financial.py"

Added Financial Record: {'record_id': 11, 'employee_id': 1, 'record_date': datetime.datetime(2024, 5, 7, 16, 27, 56, 463259), 'description': 'Bonus', 'amount':

Financial Record by ID: {'record_id': 1, 'employee_id': 101, 'record_date': datetime.datetime(2024, 5, 1, 0, 0), 'description': 'Office supplies purchase', 'am

Financial Records for Employee: [{'record_id': 11, 'employee_id': 1, 'record_date': datetime.datetime(2024, 5, 7, 16, 27, 56, 463259), 'description': 'Bonus',

Financial Records for Date: [{'record_id': 11, 'employee_id': 1, 'record_date': datetime.datetime(2024, 5, 7, 16, 27, 56, 463259), 'description': 'Bonus', 'amo

Process finished with exit code 0
```

DatabaseContext:

• A class responsible for handling database connections and interactions.

```
import mysql.connector

class DatabaseContext:
    def __init__(self, host, username, password, database):
        self.host = host
        self.username = username
        self.password = password
        self.database = database
        self.connection = None

    def connect_to_database(self):
        self.connection = mysql.connector.connect(
            host=self.host,
            user=self.username,
            password=self.password,
            database=self.database
        )
        print("Connected to the MySQL database!")
```

```python
    def execute_query(self, query):
        if not self.connection:
            self.connect_to_database()
        cursor = self.connection.cursor()
        cursor.execute(query)
        results = cursor.fetchall()
        cursor.close()
        return results
db_context = DatabaseContext(host="localhost", username="root",
password="root", database="case_study")
db_context.connect_to_database()
query = "SELECT * FROM employee"
results = db_context.execute_query(query)
print("Query Results:", results)
```

```
database
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/case study/database.py"
Connected to the MySQL database!
Query Results: []

Process finished with exit code 0
```

ValidationService:

• A class for input validation and business rule enforcement

```python
from datetime import datetime
class ValidationService:
    def validate_employee_data(self, employee_data):
        return True
ValidationService1 = ValidationService()

employee_data = {
    "EmployeeID": 101,
    "FirstName": "Ilakiya",
    "LastName": "Rangaraju",
    "DateOfBirth": "2002-10-27",
    "Gender": "Female",
    "Email": "ilakiya@gmail.com",
    "PhoneNumber": "7896543789",
    "Address": "Kottaipudhur, Erode",
    "Position": "Manager",
    "JoiningDate": "2023-01-01",
    "TerminationDate": None
```

```
}
is_valid = ValidationService1.validate_employee_data(employee_data)
if is_valid:
   print("Employee data is valid.")
else:
   print("Employee data is not valid.")
```

```
validation
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/case study/validation.py"
Employee data is valid.

Process finished with exit code 0
```

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.

```
from EmployeeService import EmployeeService
from PayrollService import PayrollService
from TaxService import TaxService
from database import DatabaseContext
print("---------")
class EmployeeNotFoundException(Exception):
   def __init__(self, employee_id):
       super().__init__(f"Employee with ID {employee_id} not found.")
class PayrollGenerationException(Exception):
   def __init__(self, employee_id, reason):
```

```
        super().__init__(f"Error generating payroll for employee {employee_id}:
{reason}")

class TaxCalculationException(Exception):
   def __init__(self, employee_id, reason):
        super().__init__(f"Error calculating taxes for employee {employee_id}:
{reason}")

class FinancialRecordException(Exception):
   def __init__(self, message):
       super().__init__(message)

class InvalidInputException(Exception):
   def __init__(self, field_name, message):
       super().__init__(f"Invalid input for field '{field_name}': {message}")

class DatabaseConnectionException(Exception):
   def __init__(self, message):
       super().__init__(f"Database connection error: {message}")

obj1=EmployeeService()
obj2=PayrollService()
obj3=TaxService()
obj4=DatabaseContext(host="localhost",     username="root",     password="root",
database="casestudy")

#1
try:
   employee = obj1.GetEmployeeById(1000)
except EmployeeNotFoundException as e:
   print(e)

# Payroll generation error
try:
   obj2.GeneratePayroll(1, "2023-10-01", "2023-10-31")
except PayrollGenerationException as e:
   print(e)

# Tax calculation error
try:
    obj3.CalculateTax(2, 2022)   # Assuming missing tax data for employee 2 in
2022
except TaxCalculationException as e:
   print(e)

# Invalid input
try:
     obj1.AddEmployee({"FirstName": "John", "Age": 30})   # Missing required
fields
```

```python
except InvalidInputException as e:
   print(e)

# Database connection issue
try:
   obj4.connect_to_database()
except DatabaseConnectionException as e:
   print(e)
```

```
---------
Failed to add employee
Connected to the MySQL database!

Process finished with exit code 0
```

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.

```
# test_service.py

import unittest
from abclass import PayrollService
from abclass import TaxService
from abclass import FinancialRecordService
from datetime import datetime

class TestPayrollService(unittest.TestCase):
   def setUp(self):
       self.db_connection = "case_study"
       self.payroll_service = PayrollService(self.db_connection)

   def test_generate_payroll(self):
       employee_id = 1
       start_date = datetime(2024, 4, 1)
       end_date = datetime(2024, 4, 15)
               payroll_id  =  self.payroll_service.generate_payroll(employee_id,
start_date, end_date)
       self.assertIsNotNone(payroll_id)

class TestTaxService(unittest.TestCase):
   def setUp(self):
       self.db_connection = "case_study"
       self.tax_service = TaxService(self.db_connection)

   def test_calculate_tax(self):
       employee_id = 1
       tax_year = 2024
       tax_id = self.tax_service.calculate_tax(employee_id, tax_year)
       self.assertIsNotNone(tax_id)

class TestFinancialRecordService(unittest.TestCase):
   def setUp(self):
       self.db_connection = "case_study"
                                             self.financial_record_service      =
FinancialRecordService(self.db_connection)

   def test_add_financial_record(self):
       employee_id = 1
       description = "Bonus"
       amount = 500
       record_type = "Income"
```

```
                                                        record_id        =
self.financial_record_service.add_financial_record(employee_id,     description,
amount, record_type)
        self.assertIsNotNone(record_id)

if __name__ == '__main__':
    unittest.main()
```

```
Tests passed: 3 of 3 tests – 0 ms
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:\PyCharm Community Edition 2021.2\plugins\python-ce\helpers\pycharm\_jb_pytest_runner.py" -
Testing started at 2:15 PM ...
H:\PyCharm Community Edition 2021.2\plugins\python-ce\helpers\pycharm\_jb_pytest_runner.py:9: DeprecationWarning: pkg_resources is deprecated as an API. See
  from pkg_resources import iter_entry_points
Launching pytest with arguments H:/case study/test.py --no-header --no-summary -q in H:\case study
H:\PyCharm Community Edition 2021.2\plugins\python-ce\helpers\pycharm\_jb_pytest_runner.py:33: DeprecationWarning: distutils Version classes are deprecated.
  elif version.LooseVersion(pytest.__version__) >= version.LooseVersion("6.0"):

============================= test session starts ==============================
collecting ... collected 3 items

test.py::TestPayrollService::test_generate_payroll PASSED               [ 33%]
test.py::TestTaxService::test_calculate_tax PASSED                      [ 66%]
test.py::TestFinancialRecordService::test_add_financial_record PASSED    [100%]

============================== 3 passed in 0.02s ===============================

Process finished with exit code 0
```

Main module

```
from datetime import datetime
from EmployeeService import EmployeeService
from PayrollService import PayrollService
from TaxService import TaxService
from financial import (FinancialRecordService)
from database import DatabaseContext

def main():
    while True:

        print("\n\n\n\n1. Get Employee by ID")
        print("2. Get All Employees")
        print("3. Add Employee")
        print("4. Update Employee")
        print("5. Remove Employee")
        print("6. Get Payroll by ID")
        print("7. Get Payrolls for Employee")
        print("8. Get Payrolls for Period")
```

```
        print("9. Calculate Tax")
        print("10. Get Tax by ID")
        print("11. Get Taxes for Employee")
        print("12. Get Taxes for Year")
        print("13. Add Financial Record")
        print("14. Get Financial Record by ID")
        print("15. Get Financial Records for Employee")
        print("16. Get Financial Records for Date")
        print("17. Database Connection")
        print("Enter 'exit' to quit.")

        choice = input("Choose an option: ")

        if choice == 'exit':
            break

        match choice:
            case '1':
                employee_service = EmployeeService()
                employee_id = int(input("Enter Employee ID: "))
                employee = employee_service.GetEmployeeById(employee_id)
                print(employee)
            case '2':
                employee_service = EmployeeService()
                employees = employee_service.GetAllEmployees()

                print(employees)
            case '3':
                employee_data = {}
                employee_service = EmployeeService()
                employee_data = {}
                employee_data["employee_id"] = input("Enter Employee ID: ")
                employee_data["first_name"] = input("Enter First Name: ")
                employee_data["last_name"] = input("Enter Last Name: ")
                    employee_data["date_of_birth"] = input("Enter Date of Birth
(YYYY-MM-DD): ")
                employee_data["gender"] = input("Enter Gender: ")
                employee_data["email"] = input("Enter Email: ")
                employee_data["phone_number"] = input("Enter Phone Number: ")
                employee_data["address"] = input("Enter Address: ")
                employee_data["position"] = input("Enter Position: ")
                      employee_data["joining_date"] = input("Enter Joining Date
(YYYY-MM-DD): ")
                    termination_date = input("Enter Termination Date (optional,
leave blank if none): ")
                                                employee_data["termination_date"]   =
datetime.strptime(termination_date,

"%Y-%m-%d") \
```

```
                if termination_date else None

                print("Employee data:", employee_data)
                employee_service.AddEmployee(employee_data)
                print("Employee added successfully!")
            case '4':
                employee_data = {}
                employee_service = EmployeeService()
                employee_service.UpdateEmployee(employee_data)
                print("Employee updated successfully!")
            case '5':
                employee_id = input("Enter Employee ID to remove: ")
                employee_service = EmployeeService()
                employee_service.RemoveEmployee(employee_id)
                print("Employee removed successfully!")
            case '6':
                payroll_service = PayrollService()
                payroll_id = int(input("Enter Payroll ID: "))
                payroll = payroll_service.GetPayrollById(payroll_id)
                print(payroll)
            case '7':
                payroll_service = PayrollService()
                employee_id = int(input("Enter Employee ID: "))
                payrolls = payroll_service.GetPayrollsForEmployee(employee_id)
                for payroll in payrolls:
                    print(payroll)
            case '8':
                payroll_service = PayrollService()
                start_date = input("Enter Start Date: ")
                end_date = input("Enter End Date: ")
                start_date = datetime.strptime(start_date, "%Y-%m-%d")
                end_date = datetime.strptime(end_date, "%Y-%m-%d")
                     payrolls = payroll_service.GetPayrollsForPeriod(start_date,
end_date)
                for payroll in payrolls:
                    print(payroll)
            case '9':
                tax_service = TaxService()
                employee_id = input("Enter Employee ID: ")
                taxYear=2020
                tax = tax_service.CalculateTax(employee_id,taxYear)
                print(tax)
            case '10':
                tax_service = TaxService()
                tax_id = int(input("Enter Tax ID: "))
                tax = tax_service.GetTaxById(tax_id)
                print(tax)
            case '11':
                tax_service = TaxService()
```

```
                employee_id = int(input("Enter Employee ID: "))
                taxes = tax_service.GetTaxesForEmployee(employee_id)
                print(taxes)
            case '12':
                tax_service = TaxService()
                tax_year = int(input("Enter Tax Year: "))
                taxes = tax_service.GetTaxesForYear(tax_year)
                print(taxes)
            case '13':
                financial_record_service = FinancialRecordService()
                employee_id = int(input("Enter Emp ID: "))
                description = str(input("Expense/Bonus/Income: "))
                amount = int(input("Enter the amount: "))
                record_type = description
                       x=financial_record_service.AddFinancialRecord(employee_id,
description, amount, record_type)
                print(x)
                print("Financial record added successfully!")
            case '14':
                financial_record_service = FinancialRecordService()
                record_id = int(input("Enter Record ID: "))
                                                        financial_record    =
financial_record_service.GetFinancialRecordById(record_id)
                print(financial_record)
            case '15':
                financial_record_service = FinancialRecordService()
                employee_id = int(input("Enter Employee ID: "))
                                                       financial_records    =
financial_record_service.GetFinancialRecordsForEmployee(employee_id)
                print(financial_records)
            case '16':
                financial_record_service = FinancialRecordService()
                record_date = input("Enter Record Date (YYYY-MM-DD): ")
                if len(record_date) == 10 and record_date.count('-') == 2:
                    try:
                                    record_date  =  datetime.strptime(record_date,
"%Y-%m-%d").date()
                                                        financial_records   =
financial_record_service.GetFinancialRecordsForDate(record_date)
                        if financial_records:
                            for record in financial_records:
                                print(record)
                        else:
                             print("No financial records found for the specified
date.")
                    except ValueError:
                              print("Invalid date format. Please enter date in
YYYY-MM-DD format.")
                else:
```

```
                    print("Invalid date format. Please enter date in YYYY-MM-DD
format.")

            case '17':
                            database_context = DatabaseContext(host="localhost",
username="root", password="root", database="case_study")
                database_context.connect_to_database()
                print("Database connection established!")

if __name__ == "__main__":
    main()
```

```
1. Get Employee by ID
2. Get All Employees
3. Add Employee
4. Update Employee
5. Remove Employee
6. Get Payroll by ID
7. Get Payrolls for Employee
8. Get Payrolls for Period
9. Calculate Tax
10. Get Tax by ID
11. Get Taxes for Employee
12. Get Taxes for Year
13. Add Financial Record
14. Get Financial Record by ID
15. Get Financial Records for Employee
16. Get Financial Records for Date
17. Database Connection
Enter 'exit' to quit.
Choose an option: 9
Enter Employee ID: 102
{'tax_id': 11, 'employee_id': '102', 'tax_year': 2020, 'taxable_income': 60000, 'tax_amount': 12000}
```