CASE STUDY

PAYEXPERT

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

    @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

    @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

    @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=1,
    first_name="Logesh",
    last_name="Dhamodaran",
    date_of_birth=datetime(2002, 10, 22),
    gender="Male",
    email="logesh@example.com",
    phone_number="56738200",
    address="12 Metro,Cbe",
    position="Manager",
    joining_date=datetime(2020, 1, 1),
    termination_date=None)
print("AGE IS : ",Employee1.calculate_age())
```

OUTPUT:

```
C:\Users\Sathish\PycharmProjects\pythonProjectcs\.
AGE IS :  21

Process finished with exit code 0
```

EmployeeService (implements IEmployeeService):

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

```python
from abs_class import IEmployeeService
from datetime import datetime
class EmployeeService(IEmployeeService):
    def __init__(self):
        self.employee_data = {}

    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
            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": 1,
    "first_name": "Logesh",
    "last_name": "Dhamodaran",
    "date_of_birth": datetime(2002, 10, 22),
    "gender": "Male",
    "email": "logesh@example.com",
    "phone_number": "1234567890",
    "address": "123 Main St, City",
    "position": "Manager",
    "joining date": datetime(2020, 1, 1),
    "termination_date": None
}
```

```python
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)

#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, 1, 1),
    "termination_date": None
}
Employee_service1.UpdateEmployee(updated_employee_data)

#5
Employee_service1.RemoveEmployee(4)
```

OUTPUT:

```
Employee not found
Employee added successfully
All Employees:
Employee ID: 1
First Name: Logesh
Last Name: Dhamodaran
Employee updated successfully
Failed to remove employee

Process finished with exit code 0
```

PayrollService (implements IPayrollService):

- Methods:
  - GeneratePayroll
  - GetPayrollById
  - GetPayrollsForEmployee
  - GetPayrollsForPeriod

```
from datetime import datetime
from abs_class import IPayrollService

class PayrollService(IPayrollService):
    def __init__(self):
        self.payroll_data = {}

    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


Payroll_service1 = PayrollService()

#1
employee_id = 1
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)
```

OUTPUT:

```
Generated Payroll:  {'payroll_id': 1, 'employee_id': 1, 'pay_period_start_date': datetime.datetime(2024, 4, 1, 0, 0), 'pay_period_end_date': datetime.datetime(

Payroll by ID: {'payroll_id': 1, 'employee_id': 1, 'pay_period_start_date': datetime.datetime(2024, 4, 1, 0, 0), 'pay_period_end_date': datetime.datetime(2024,

Payrolls for Employee: [{'payroll_id': 1, 'employee_id': 1, 'pay_period_start_date': datetime.datetime(2024, 4, 1, 0, 0), 'pay_period_end_date': datetime.datet

Payrolls within Period: [{'payroll_id': 1, 'employee_id': 1, 'pay_period_start_date': datetime.datetime(2024, 4, 1, 0, 0), 'pay_period_end_date': datetime.date

Process finished with exit code 0
```

TaxService (implements ITaxService):

- Methods:
  - CalculateTax
  - GetTaxById
  - GetTaxesForEmployee

```python
from datetime import datetime
from abs_class import ITaxService
class TaxService(ITaxService):
    def __init__(self):
       self.tax_data = {}

    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)
```

```python
        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)
```

OUTPUT:

```
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 abs_class import IFinancialRecordService
class FinancialRecordService(IFinancialRecordService):
    def __init__(self):
        self.financial_records = {}
    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.date():
                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("Added Financial Record:", added_record)
#2
record_id = 1
record_by_id =
FinancialRecordSservice1.GetFinancialRecordById(record_id)
print("Financial Record by ID:", record_by_id)
```

```
#3
records_for_employee =
FinancialRecordSservice1.GetFinancialRecordsForEmployee(employee_
id)
print("Financial Records for Employee:", records_for_employee)
#4
record_date = datetime.now().date()
records_for_date =
FinancialRecordSservice1.GetFinancialRecordsForDate(record_date)
print("Financial Records for Date:", records_for_date)
```

OUTPUT:

```
Added Financial Record: {'record_id': 1, 'employee_id': 1, 'record_date': datetime.datetime(2024, 5, 6, 16, 17, 30, 94913), 'description': 'Bonus', 'amount': 1(

Financial Record by ID: {'record_id': 1, 'employee_id': 1, 'record_date': datetime.datetime(2024, 5, 6, 16, 17, 30, 94913), 'description': 'Bonus', 'amount': 1(

Financial Records for Employee: [{'record_id': 1, 'employee_id': 1, 'record_date': datetime.datetime(2024, 5, 6, 16, 17, 30, 94913), 'description': 'Bonus', 'a

Financial Records for Date: [{'record_id': 1, 'employee_id': 1, 'record_date': datetime.datetime(2024, 5, 6, 16, 17, 30, 94913), 'description': 'Bonus', 'amoun
```

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!")
```

```
    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="casestudy")
db_context.connect_to_database()
query = "SELECT * FROM employee"
results = db_context.execute_query(query)
print("Query Results:", results)
```

OUTPUT:

```
C:\Users\Sathish\PycharmProjects\python
Connected to the MySQL database!
Query Results: []

Process finished with exit code 0
```

ValidationService:

• A class for input validation and business rule enforcement.

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

employee_data = {
    "EmployeeID": 1,
    "FirstName": "John",
    "LastName": "Doe",
    "DateOfBirth": "1990-01-01",
    "Gender": "Male",
    "Email": "john.doe@example.com",
    "PhoneNumber": "1234567890",
    "Address": "123 Main Street",
    "Position": "Manager",
    "JoiningDate": "2020-01-01",
```

```
    "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.")
```

OUTPUT:

```
C:\Users\Sathish\PycharmProjects\pythonProjectcs
Employee data is valid.

Process finished with exit code 0
```

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)

```
from abc import ABC, abstractmethod

class IEmployeeService(ABC):
    @abstractmethod
    def GetEmployeeById(self, employeeId):
        pass

    @abstractmethod
    def GetAllEmployees(self):
        pass

    @abstractmethod
    def AddEmployee(self, employeeData):
        pass

    @abstractmethod
    def UpdateEmployee(self, employeeData):
        pass

    @abstractmethod
    def RemoveEmployee(self, employeeId):
        pass

class IPayrollService(ABC):
    @abstractmethod
    def GeneratePayroll(self, employeeId, startDate, endDate):
        pass

    @abstractmethod
    def GetPayrollById(self, payrollId):
        pass

    @abstractmethod
    def GetPayrollsForEmployee(self, employeeId):
        pass

    @abstractmethod
```

```
    def GetPayrollsForPeriod(self, startDate, endDate):
        pass

class ITaxService(ABC):
    @abstractmethod
    def CalculateTax(self, employeeId, taxYear):
        pass

    @abstractmethod
    def GetTaxById(self, taxId):
        pass

    @abstractmethod
    def GetTaxesForEmployee(self, employeeId):
        pass

    @abstractmethod
    def GetTaxesForYear(self, taxYear):
        pass

class IFinancialRecordService(ABC):
    @abstractmethod
    def AddFinancialRecord(self, employeeId, description, amount,
recordType):
        pass

    @abstractmethod
    def GetFinancialRecordById(self, recordId):
        pass

    @abstractmethod
    def GetFinancialRecordsForEmployee(self, employeeId):
        pass

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

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 file2 import EmployeeService
from file3 import PayrollService
from file4 import TaxService
from dbconn 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()
```

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

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

OUTPUT:

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

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.

```
import unittest
import pytest
#from pythonProjectcs.services import EmployeeService,
PayrollService
from file2 import EmployeeService
from file3 import PayrollService
@pytest.mark.parametrize("basic_salary, overtime_pay,
expected_gross_salary", [
    (10000, 500, 10500),
    (15000, 1000, 16000),
])
def test_calculate_gross_salary_for_employee(basic_salary,
overtime_pay, expected_gross_salary):
    # Arrange
    employee_id = 1
    employee_service = EmployeeService()
    employee = employee_service.GetEmployeeById(employee_id)

    # Act
```

```
    gross_salary = PayrollService.CalculateGrossSalary(employee,
basic_salary, overtime_pay)

    # Assert
    assert gross_salary == expected_gross_salary
```

OUTPUT: