Low-Level Design (LLD) for Employee Dictionary Operations

Difficulty Level: Medium | Total Marks: 10

Standards Followed: 3 Functions | 3 Visible Test Cases

Concepts Tested

- Nested dictionary operations
- Dictionary traversal using items(), keys(), values()
- Conditional statements (if-else)
- Aggregation and filtering logic
- Searching and validating dictionary keys
- Basic data processing using loops

==========

Problem Statement

Design a system that manages a dictionary of employee records and performs operations such as:

- Calculating the average salary of employees belonging to the IT department
- Finding the employee with the highest salary across all departments
- Checking whether a specific employee ID exists and displaying employee details

Given Input:

```
employees = {
    "E001": {"name": "Raj", "department": "IT", "salary": 50000},
    "E002": {"name": "Priya", "department": "HR", "salary": 45000},
    "E003": {"name": "Amit", "department": "IT", "salary": 55000},
    "E004": {"name": "Sara", "department": "Finance", "salary": 48000},
    "E005": {"name": "Vikram", "department": "IT", "salary": 60000}
}
```

==========

Operations

1. Calculate Average IT Department Salary

Function Prototype:
def calculate_average_it_salary(self):

Expected Output:
Average IT Salary: 55000.0

---

2. Find Highest Paid Employee

Function Prototype:
def find_highest_paid_employee(self):

Expected Output:
Highest Paid Employee: Vikram (E005) - 60000

---

3. Check Employee Existence by ID

Function Prototype:
def check_employee(self, emp_id):

Expected Output:
Employee E002: Priya - 45000

==========

Implementation Code

```
class EmployeeManager:

    def __init__(self):
        """Initialize employee dictionary."""
        self.employees = {
            "E001": {"name": "Raj", "department": "IT", "salary": 50000},
            "E002": {"name": "Priya", "department": "HR", "salary": 45000},
            "E003": {"name": "Amit", "department": "IT", "salary": 55000},
            "E004": {"name": "Sara", "department": "Finance", "salary": 48000},
            "E005": {"name": "Vikram", "department": "IT", "salary": 60000}
        }

    def calculate_average_it_salary(self):
        """Calculate and print average IT salary."""
        total = 0
        count = 0
```

```
        for emp in self.employees.values():
            if emp["department"] == "IT":
                total += emp["salary"]
                count += 1
        average = total / count if count > 0 else 0
        print("Average IT Salary:", average)

    def find_highest_paid_employee(self):
        """Find and print highest paid employee."""
        highest_id = None
        highest_data = None
        for emp_id, emp in self.employees.items():
            if highest_data is None or emp["salary"] > highest_data["salary"]:
                highest_id = emp_id
                highest_data = emp
        print(f"Highest Paid Employee: {highest_data['name']} ({highest_id}) -
{highest_data['salary']}")

    def check_employee(self, emp_id):
        """Check employee existence and print details."""
        emp = self.employees.get(emp_id)
        if emp:
            print(f"Employee {emp_id}: {emp['name']} - {emp['salary']}")
        else:
            print(f"Employee {emp_id} not found")
```

Test Case Table

—----------

Test Case ID	Test Case Description	Associated Function(s)	Marks

---

-

TC1 | Calculate average IT department salary | calculate_average_it_salary() | 3 Marks

TC2 | Find highest paid employee | find_highest_paid_employee() | 3 Marks

TC3 | Check employee existence by ID | check_employee() | 4 Marks

TOTAL | All test cases passed | - | 10 Marks

==========

Visible Test Cases

TC1 Input:

1

average_it

Output:

Average IT Salary: 55000.0

TC2 Input:

1

highest

Output:

Highest Paid Employee: Vikram (E005) - 60000

TC3 Input:

1

check E002

Output:
Employee E002: Priya - 45000