
CASE STUDY 1

Smart University Management System (Python OOP Based)

Smart University Management System (Python OOP Based)

Background

A university wants to develop a **Python-based management system** to handle students, faculty, courses, and evaluations using **Object-Oriented Programming principles**.

Problem Statement

Design and implement a **Smart University Management System** using Python OOP concepts to manage academic operations efficiently.

System Requirements

1. **Introduction to OOP**
    - Use classes and objects to model real-world entities.
2. **Classes & Objects**
    - Create classes:
        - `Person`
        - `Student`
        - `Faculty`
        - `Course`
        - `Department`
3. **Constructors & Destructors**
    - Initialize objects using constructors.

- Log cleanup actions using destructors.

4. **Parameterized Methods**
   - Methods to:
     - Enroll students
     - Assign faculty
     - Calculate grades
5. **Types of Classes**
   - Abstract base classes
   - Utility/helper classes
6. **Inheritance**
   - `Student` and `Faculty` inherit from `Person`
7. **Types of Inheritance**
   - Single inheritance
   - Multilevel inheritance
   - Hierarchical inheritance
8. **Polymorphism**
   - Method overriding for:
     - `get_details()`
     - `calculate_performance()`
9. **Operator Overloading**
   - Overload operators:
     - `+` to merge course credits
     - `>` to compare student performance
10. **Descriptors**
    - Implement descriptors to:

        - Validate marks (0–100)
        - Control salary access

11. **Decorators**
    - Create decorators for:
        - Access control (Admin only)
        - Logging method execution
        - Performance timing
12. **Iterators & Generators**
    - Generator to yield student records batch-wise
    - Iterator for course traversal
13. **File Handling**
    - Store student data in:
        - JSON files
        - CSV reports
    - Read/write/update records
14. **Exception Handling**
    - Handle invalid data entry
    - Duplicate records
    - File access errors

Expected Outcome

- Modular, extensible OOP-based system
- Clear demonstration of all OOP principles
- Real-world simulation of university operations

Input Format

All inputs are provided **either through console input or from files (JSON/CSV)**.

---

1. Student Details Input

```
Student ID
Student Name
Department
Semester
Marks (5 subjects separated by space)
```

Example Input

```
S101
Ananya Sharma
Computer Science
4
78 85 90 88 92
```

---

2. Faculty Details Input

```
Faculty ID
Faculty Name
Department
Monthly Salary
```

Example Input

```
F201
Dr. Rajesh Kumar
Computer Science
85000
```

---

3. Course Details Input

```
Course Code
Course Name
Credits
Faculty ID
```

Example Input

```
CS401
Data Structures
4
F201
```

---

4. User Choice (Menu Driven Input)

```
1 → Add Student
2 → Add Faculty
3 → Add Course
4 → Enroll Student to Course
5 → Calculate Student Performance
6 → Compare Two Students
7 → Generate Reports
8 → Exit
```

Example Input

```
1
```

---

5. File-Based Input (JSON / CSV)

students.json

```
[
  {
    "id": "S101",
    "name": "Ananya Sharma",
    "department": "Computer Science",
    "semester": 4,
    "marks": [78, 85, 90, 88, 92]
  }
]
```

Output Format

---

1. Student Creation Output

```
Student Created Successfully
--------------------------------
ID         : S101
Name       : Ananya Sharma
Department: Computer Science
Semester   : 4
```

---

2. Faculty Creation Output

```
Faculty Created Successfully
--------------------------------
ID         : F201
Name       : Dr. Rajesh Kumar
Department: Computer Science
```

---

3. Course Creation Output

```
Course Added Successfully
--------------------------------
Course Code : CS401
Course Name : Data Structures
Credits     : 4
Faculty     : Dr. Rajesh Kumar
```

---

4. Student Enrollment Output

```
Enrollment Successful
--------------------------------
Student Name : Ananya Sharma
Course       : Data Structures
```

---

5. Student Performance Calculation Output

```
Student Performance Report
--------------------------------
Student Name : Ananya Sharma
Marks        : [78, 85, 90, 88, 92]
Average      : 86.6
Grade        : A
```

(Average calculated using generator / iterator)

---

6. Polymorphism Output (Method Overriding)

```
Student Details:
--------------------------------
Name      : Ananya Sharma
Role      : Student
Department: Computer Science
Faculty Details:
--------------------------------
Name      : Dr. Rajesh Kumar
Role      : Faculty
Department: Computer Science
```

---

7. Operator Overloading Output

Compare Two Students (> operator)

```
Comparing Students Performance
--------------------------------
Ananya Sharma > Rohan Verma : True
```

Merge Course Credits (+ operator)

```
Total Credits After Merge : 7
```

---

8. Descriptor Validation Output

Invalid Marks

```
Error: Marks should be between 0 and 100
```

Unauthorized Salary Access

```
Access Denied: Salary is confidential
```

---

9. Decorator Output (Logging / Access Control)

```
[LOG] Method calculate_performance() executed successfully
Access Denied: Admin privileges required
```

---

10. Iterator / Generator Output

Student Record Generator

```
Fetching Student Records...
--------------------------------
S101 - Ananya Sharma
S102 - Rohan Verma
```

---

11. File Output

CSV Report (students_report.csv)

```
ID,Name,Department,Average,Grade
S101,Ananya Sharma,Computer Science,86.6,A
```

JSON Output Confirmation

```
Student data successfully saved to students.json
```

---

12. Exception Handling Output

```
Error: Student ID already exists
Error: File not found
```

---

13. Exit Output

```
Thank you for using Smart University Management System
```
