🔥 **DAY 2 — OOP KA "SECURITY + PROFESSIONALISM" LAYER** *(Encapsulation • @property • Custom Exceptions) (Same style • Same depth • Same "design first" thinking — bas level aur upar)*

Bro, agar **Day 1 = Ghar ka structure**, to **Day 2 = Ghar ka lock, rules aur safety system** 🔐

Aaj ka goal:

> **"Code ko kaam karwana hi nahi, balki galat use se bachana."**

Yehi cheez **real backend, payments, auth, AI scoring systems** me sabse zyada kaam aati hai.

🧠 DAY 2 MINDSET (VERY IMPORTANT)

❌ Beginner thinking

> "User jo bhi bheje, accept kar lo"

☑️ Professional thinking

> **"User se kuch bhi aa sakta hai — mujhe system ko safe rakhna hai."**

👉 Isi thinking se aata hai:

- Encapsulation
- Validation
- Exceptions

1️⃣ ENCAPSULATION — DATA PROTECTION

❓ Sabse pehle sawaal:

Encapsulation ki zarurat kyun padi?

Socho ye situation 👇

```
s1 = Student("Arun", 78)
s1.marks = -500    # 😳 allowed?
```

❌ Ye **bahut dangerous** hai Backend / AI / Exams / Payments — sab toot jaate hain.

👉 **Solution = Encapsulation**

◇ Encapsulation kya hota hai? (Feynman style)

> **Encapsulation = Data ko seedha access se bachana aur uske liye controlled gate banana**

Real-life example 🏧

- ATM machine
- Tum seedha bank ke server ko touch nahi kar sakte
- Sirf **ATM interface** se kaam hota hai

◇ Python me Encapsulation kaise hota hai?

Python me 3 levels hote hain:

Level	Syntax	Matlab
Public	`marks`	Sab access kar sakte
Protected	`_marks`	“Samajhdaar log hi use kare”
Private	`__marks`	Class ke bahar almost nahi

◇ PRIVATE VARIABLES (`__variable`)

```
class Student:
    def __init__(self, marks):
        self.__marks = marks
```

Ab ye kaam nahi karega:

```
s1.__marks    # ❌ error
```

🧠 Python internally naam badal deta hai `__marks → _Student__marks`

🔥 IMPORTANT TRUTH (INTERVIEW GOLD)

> Python me **true private kuch nahi hota**, par **intent clear hota hai**: “Isse bahar se mat chhedo.”

---

2 GETTERS & SETTERS — CONTROLLED ACCESS

Encapsulation ka matlab **data chhupa dena** nahi, balke **safe tareeke se dena**.

◇ Getter (read access)

```
class Student:
    def __init__(self, marks):
        self.__marks = marks

    def get_marks(self):
        return self.__marks
```

◇ Setter (write access with rules)

```
    def set_marks(self, marks):
        if marks < 0:
            print("Invalid marks")
        else:
            self.__marks = marks
```

Usage:

```
s1.set_marks(90)
```

❌ PROBLEM with traditional getter/setter

Code ugly ho jata hai:

```
s1.get_marks()
s1.set_marks(80)
```

👉 Python ka solution = **@property** 😎

3️⃣ @property — PYTHON KA SUPERPOWER

❓ @property kyun aaya?

> **Taaki code dikhe simple par control rahe full**

🔹 Basic example

```
class Student:
    def __init__(self, marks):
        self.__marks = marks

    @property
    def marks(self):
        return self.__marks
```

Access:

```
print(s1.marks)    # looks like variable
```

But actually 👉 **method chal rahi hai**

🔹 Setter with @property

```
class Student:
    def __init__(self, marks):
        self.__marks = marks

    @property
    def marks(self):
        return self.__marks
    @marks.setter
    def marks(self, value):
        if value < 0:
            raise ValueError("Marks cannot be negative")
        self.__marks = value
```

Now:

```
s1.marks = 85      # safe
s1.marks = -10    # ❌ error
```

🔥 **This is real professional OOP**

🧠 Visualization

```
Outside World
     |
     v
 s1.marks  --->  @property method  --->  __marks
```

4️⃣ CUSTOM EXCEPTIONS — SYSTEM KO STRONG BANAO

❓ Normal error enough kyun nahi?

```
ValueError
TypeError
```

Ye generic hote hain.

Real systems me chahiye:

- InvalidScoreError
- PaymentFailedError
- UnauthorizedUserError

👉 **Readable + Debuggable**

◇ Custom Exception banana

```python
class InvalidMarksError(Exception):
    pass
```

Use it:

```python
if marks < 0:
    raise InvalidMarksError("Marks cannot be negative")
```

◇ Full example (REAL WORLD STYLE)

```python
class InvalidMarksError(Exception):
    pass


class Student:
    def __init__(self, marks):
        self.__marks = marks

    @property
    def marks(self):
        return self.__marks

    @marks.setter
    def marks(self, value):
        if value < 0 or value > 100:
            raise InvalidMarksError("Marks must be 0-100")
        self.__marks = value
```

🔥 WHY THIS MATTERS IN BACKEND / AI

- API response me clear error
- Logs readable
- Debugging easy
- User ko proper message

FastAPI loves this 💚

🛠️ PRACTICE — SMALL BUT COMPLETE SYSTEM

🎯 Mini Project (Day 2)

Requirement:

- Student has private marks
- Safe update via property
- Invalid input throws custom error

```python
class InvalidMarksError(Exception):
    pass


class Student:
    def __init__(self, name, marks):
        self.name = name
        self.__marks = marks

    @property
    def marks(self):
        return self.__marks

    @marks.setter
    def marks(self, value):
        if value < 0 or value > 100:
            raise InvalidMarksError("Invalid marks")
        self.__marks = value
```

Test:

```python
s1 = Student("Arun", 78)
s1.marks = 95      # ✅
s1.marks = -20     # ❌
```

🔑 KEY INSIGHTS (SECRET SAUCE)

1️⃣ **Har data public mat rakho** Especially: score, money, auth

2️⃣ **@property use karo jab:**

- validation chahiye
- future me logic add ho sakta hai

3️⃣ **Custom exceptions = mature developer sign**

🤖 AI ERA ME ISKA USE

AI systems me:

- Wrong input = garbage output
- Encapsulation ensures **clean data**
- Exceptions ensure **safe failure**

LLM pipelines me:

- Prompt size
- Token limits
- Confidence scores

👉 sab protected hote hain

🧠 QUICK REVISION (PACED REPETITION)

Answer bina dekhe:

1. Encapsulation kya hota hai?
2. __variable ka matlab?
3. @property kyun better hai getter/setter se?
4. Custom exception kyun banate hain?
5. marks ko direct access kyun dangerous hai?

Agar ye clear hai → 🔥 **Day 2 bhi DONE**

🚀 NEXT DAY PREVIEW — DAY 3

- Inheritance
- super()

- Polymorphism
- Real exam system extension

Bhai ready ho jao — **kal system “expand” karna seekhenge** 💥