Comprehensive Mastery of Object-Oriented Programming (OOP)

Day 4: Comprehensive Mastery of Object-Oriented Programming (OOP)

Learning Objectives

By the end of Day 4, students will:
- **Understand the OOP pillars**: Classes, objects, inheritance, polymorphism, encapsulation, abstraction.
- **Develop practical applications** with **at least 15 hands-on projects**. - Learn **design patterns** for efficient software design. - Build **real-world systems**, such as **Hospital Management**, **Bank Systems**, and **Library Management** using OOP principles.

---

1. Introduction to Classes and Objects (60 minutes)

What are Classes and Objects?

- **Class**: A template for creating objects.
- **Object**: An instance of a class, representing a specific entity with data and behavior.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"My name is {self.name} and I am {self.age} years old.")

# Create an object
person1 = Person("Alice", 25)
person1.introduce()
```

Hands-on Exercise:

1. Create a **`Product` class** with attributes for product name, price, and stock.
2. Add a **method to print product details**.

---

2. Project 1: Employee Record System (30 minutes)

Goal:
Create a system that stores and displays employee records using OOP.

```
class Employee:
    def __init__(self, name, position, salary):
        self.name = name
        self.position = position
        self.salary = salary

    def display_info(self):
        print(f"Employee: {self.name}, Position: {self.position}, Salary:
{self.salary}")

# Example usage
emp1 = Employee("John", "Manager", 50000)
emp2 = Employee("Sarah", "Developer", 60000)
emp1.display_info()
emp2.display_info()
```

---

3. Inheritance – Reusability of Code (60 minutes)

Single and Multiple Inheritance

```
class Vehicle:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def start(self):
        print(f"{self.brand} {self.model} starts.")

class Car(Vehicle):
    def __init__(self, brand, model, seats):
        super().__init__(brand, model)
        self.seats = seats

    def honk(self):
        print(f"{self.brand} honks!")

car = Car("Toyota", "Corolla", 5)
car.start()
car.honk()
```

Project 2: Vehicle Inheritance System (30 minutes)

- Create a **`Motorcycle` class** inheriting from `Vehicle`.
- Add a method to display whether it has a **sidecar**.

4. Polymorphism – Flexibility in Behavior (60 minutes)

```
class Animal:
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        print("Bark")

class Cat(Animal):
    def sound(self):
        print("Meow")

animals = [Dog(), Cat()]
for animal in animals:
    animal.sound()
```

Project 3: Shape Area Calculator (45 minutes)

- Create a **base `Shape` class** with an `area()` method.
- Implement **`Circle`**, **`Rectangle`**, and **`Triangle`** classes with specific formulas.

```
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

circle = Circle(5)
print(f"Circle Area: {circle.area()}")
```

---

5. Encapsulation – Data Hiding (45 minutes)

```
class BankAccount:
    def __init__(self, owner, balance=0):
        self.__balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
```

```
    def get_balance(self):
        return self.__balance

account = BankAccount("Alice", 100)
account.deposit(50)
print(account.get_balance())  # Outputs: 150
```

Project 4: Password Manager (40 minutes)

- Implement a **`PasswordManager` class** with private storage for passwords.
- Add methods to **add, update, and retrieve passwords**.

---

6. Abstraction – Simplifying Interfaces (45 minutes)

```
from abc import ABC, abstractmethod

class Payment(ABC):
    @abstractmethod
    def process_payment(self, amount):
        pass

class CreditCardPayment(Payment):
    def process_payment(self, amount):
        print(f"Processing credit card payment of ${amount}.")

payment = CreditCardPayment()
payment.process_payment(100)
```

Project 5: Payment System Simulation (45 minutes)

- Implement **`PayPalPayment`** as another subclass of `Payment`.
- Simulate multiple payment methods for different customers.

---

7. Capstone Project: Hospital Management System (90 minutes)

Goal

Develop a **Hospital Management System** where: 1. **Doctors and patients** are represented as objects. 2. Doctors can **schedule appointments**. 3. Patients can **view their medical history**.

Code Skeleton:

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Doctor(Person):
    def __init__(self, name, age, specialty):
        super().__init__(name, age)
        self.specialty = specialty
        self.appointments = []

    def schedule_appointment(self, patient):
        self.appointments.append(patient)

class Patient(Person):
    def __init__(self, name, age):
        super().__init__(name, age)
        self.history = []

    def add_record(self, record):
        self.history.append(record)
```

---

8. Design Patterns: Singleton & Factory (60 minutes)

Singleton Pattern Example:

```python
class Singleton:
    _instance = None

    def __new__(cls):
        if not cls._instance:
            cls._instance = super().__new__(cls)
        return cls._instance
```

Factory Pattern Example:

```python
class AnimalFactory:
    def create_animal(self, type):
        if type == "dog":
            return Dog()
        elif type == "cat":
            return Cat()
```

---

9. Recap and Homework (15 minutes)

Recap

- Covered **OOP pillars**: Classes, inheritance, polymorphism, encapsulation, abstraction.

- Completed **15+ mini-projects**.

Homework

1. Extend the **Hospital System** to include patient billing.
2. Build a **Library System** with book borrowing, returning, and overdue tracking.

Day 4 Summary

- **15+ detailed projects** completed.
- Full mastery of **OOP principles**.
- Ready for **Day 5**: Integration with **APIs and Data Analysis Tools**.