IE221 - BÀI THỰC HÀNH 4
Thực hành thiết kế hướng đối tượng trong Python

Ngày 24 tháng 10 năm 2025

Mục lục

1. NHẮC LẠI LÝ THUYẾT VÀ KHÁI NIỆM

1.1 Giới thiệu về OOP (Object-Oriented Programming)

1.1.1 Định nghĩa

Lập trình hướng đối tượng (OOP) là phương pháp lập trình tổ chức chương trình thành các đối tượng (objects) – mỗi đối tượng biểu diễn một thực thể trong thế giới thực, có dữ liệu (thuộc tính) và hành vi (phương thức) riêng.

OOP giúp lập trình viên mô hình hóa vấn đề theo cách tự nhiên, dễ bảo trì và mở rộng hơn so với lập trình thủ tục.

1.1.2 Tại sao sử dụng OOP

- Giúp tổ chức code rõ ràng, dễ hiểu theo cấu trúc lớp và đối tượng.
- Dễ mở rộng, bảo trì và tái sử dụng nhờ tính kế thừa và đa hình.
- Giảm trùng lặp code bằng cách gom nhóm dữ liệu và hành vi liên quan.
- Hỗ trợ thiết kế phần mềm lớn thông qua phân lớp, module, và mô hình hóa thực thể.

1.2 Các khái niệm cơ bản

1.2.1 Class (Lớp)

Là bản thiết kế (blueprint) mô tả cấu trúc và hành vi của đối tượng.

Ví dụ:

```
    class Student:
        def __init__(self, name, mssv):
            self.name = name
            self.mssv = mssv

        def greet(self):
            print(f"Xin chào, tôi là {self.name}, MSSV {self.mssv}")
```

1.2.2 Object (Đối tượng)

Là instance cụ thể của một class. Ví dụ:

```
    sv1 = Student("An", "123456")
    sv1.greet()  # Gọi phương thức của đối tượng
```

1.2.3 Attributes (Thuộc tính)

Là dữ liệu mô tả đặc điểm của đối tượng (như tên, mã số,...). Được khai báo trong constructor hoặc trực tiếp trong class.

1.2.4 Methods (Phương thức)

Là các hàm mô tả hành vi của đối tượng như greet() trong ví dụ trên.

1.3 Các nguyên lý cơ bản của OOP

1.3.1 Encapsulation (Đóng gói)

Đóng gói là ẩn dữ liệu nội bộ của lớp, chỉ cho phép truy cập thông qua các phương thức công khai.

```
    class BankAccount:
        def __init__(self, account_number, balance):
            self.account_number = account_number  # Public
            self._owner = ""                      # Protected (convention)
            self.__balance = balance              # Private (name mangling)

        # Getter method
        def get_balance(self):
            return self.__balance
```

```
        # Setter method với validation
        def deposit(self, amount):
            if amount > 0:
                self.__balance += amount
                return f"Deposited {amount}. New balance: {self.__balance}"
            return "Invalid amount"

        def withdraw(self, amount):
            if 0 < amount <= self.__balance:
                self.__balance -= amount
                return f"Withdrew {amount}. New balance: {self.__balance}"
            return "Insufficient funds or invalid amount"

        # Property decorator - truy cập như attribute
        @property
        def balance(self):
            return self.__balance

    # Test
    account = BankAccount("123456", 1000)
    print(account.deposit(500))      # Deposited 500. New balance: 1500
    print(account.withdraw(300))     # Withdrew 300. New balance: 1200
    print(account.balance)           # 1200 (dùng property)
    # print(account.__balance)       # AttributeError - không truy cập trực tiếp được
```

1.3.2 Inheritance (Kế thừa)

Là khả năng tạo lớp con (child class) từ lớp cha (parent class) để tái sử dụng và mở rộng mã nguồn.

```
    class Person:
        def __init__(self, name):
            self.name = name
        def introduce(self):
            print(f"Tôi là {self.name}")

    class Student(Person):   # Kế thừa Person
        def __init__(self, name, mssv):
            super().__init__(name)     # Gọi constructor lớp cha
            self.mssv = mssv
        def introduce(self):
            print(f"Sinh viên {self.name}, MSSV {self.mssv}")
```

1.3.3 Polymorphism (Đa hình)

Đa hình cho phép cùng một tên phương thức có nhiều cách thực hiện khác nhau tùy loại đối tượng.

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

    class Dog(Animal):
        def speak(self):
            return "Gâu gâu!"

    class Cat(Animal):
        def speak(self):
            return "Meo meo!"

    for a in [Dog(), Cat()]:
        print(a.speak())  # Cùng gọi speak(), kết quả khác nhau
```

1.3.4 Abstraction (Trừu tượng hóa)

Tính trừu tượng giúp loại bỏ những thứ phức tạp không cần thiết của đối tượng và chỉ tập trung vào những thứ cốt lõi quan trọng.

```
    from abc import ABC, abstractmethod

    class Shape(ABC):
        @abstractmethod
```

```
        def area(self):
            pass

    class Rectangle(Shape):
        def __init__(self, w, h):
            self.w = w
            self.h = h
        def area(self):
            return self.w * self.h
```

2. BÀI TẬP DEMO TẠI LỚP

2.1 Demo 1 – Xây dựng Hệ thống Quản lý Thư viện Sách

2.1.1 Phân tích bài toán và xác định đối tượng

Bài toán yêu cầu quản lý danh sách sách trong một thư viện – bao gồm các thao tác: thêm, xoá, tìm kiếm, mượn, trả và thống kê.
Đặt các câu hỏi phân tích:

- **Đâu là thực thể chính trong hệ thống?** ⇒ Là `Book` (Sách).
- **Ai quản lý các sách đó?** ⇒ Là `Library` (Thư viện).

Từ đó rút ra:

- `Book`: Đối tượng dữ liệu đơn lẻ, biểu diễn một cuốn sách cụ thể.
- `Library`: Đối tượng quản lý, chứa và thao tác với nhiều đối tượng `Book`.

2.1.2 Thiết kế các lớp và mối quan hệ

Lớp	Vai trò	Thuộc tính chính	Phương thức chính	Ghi chú
`Book`	Đại diện cho một cuốn sách trong thư viện.	`title, author, isbn, _price, is_available`	`display_info(), borrow(), return_book()`	`_price` được đặt private để minh họa đóng gói dữ liệu.
`Library`	Quản lý danh sách các đối tượng `Book`.	`name, _books` (danh sách)	`add_book(), remove_book(), find_book(), display_all_books()`	Thể hiện mối quan hệ *composition* (thư viện chứa nhiều sách).

Bảng 1: Thiết kế lớp trong hệ thống quản lý thư viện

Quan hệ giữa hai lớp là dạng **"has-a"** (`Library` có nhiều `Book`).

2.1.3 Gợi ý cách làm

1. **Bước 1 – Thiết kế lớp `Book`** Viết hàm `__init__`, các thuộc tính và phương thức cơ bản.
2. **Bước 2 – Thiết kế lớp `Library`** Cần danh sách `Book` để thể hiện *composition*. Cài đặt các hàm thêm, xoá, tìm kiếm và in danh sách.
3. **Bước 3 – Demo sử dụng** Tạo vài cuốn sách, thêm vào thư viện, mượn và trả. Quan sát thay đổi trạng thái `is_available`.

2.2 Demo 2 – Xây dựng Hệ thống Thanh toán Đa hình thức

2.2.1 Phân tích bài toán

Bài toán yêu cầu mô hình hóa quá trình thanh toán linh hoạt với nhiều hình thức (Credit Card, PayPal, Bank Transfer, Digital Wallet...).
Các điểm chung và khác biệt:

- **Giống nhau:** Đều có số tiền (`amount`), mô tả (`description`), các thao tác `validate()` và `process_payment()`.
- **Khác nhau:** Cách xác thực và xử lý thanh toán của từng loại khác nhau.

2.2.2 Thiết kế lớp và mối quan hệ kế thừa

Lớp	Vai trò	Ghi chú
`PaymentMethod`	Lớp trừu tượng định nghĩa giao diện chung cho tất cả các hình thức thanh toán.	Khai báo với `abc.ABC`, bắt buộc các lớp con phải triển khai `validate()` và `process_payment()`.
`CreditCard, PayPal, BankTransfer, DigitalWallet`	Các lớp con kế thừa từ `PaymentMethod`, mỗi lớp có logic xử lý và kiểm tra hợp lệ riêng.	Ví dụ: `CreditCard` kiểm tra số thẻ và CVV, `PayPal` kiểm tra email, `DigitalWallet` kiểm tra số điện thoại và PIN.
`PaymentProcessor`	Lớp điều phối chịu trách nhiệm gọi `process_payment()` và lưu lịch sử giao dịch.	Minh họa cho tính đa hình (*polymorphism*) — cùng một lời gọi hàm hoạt động khác nhau tùy đối tượng.

Bảng 2: Thiết kế lớp trong hệ thống thanh toán đa hình thức

Quan hệ:

- Các lớp `CreditCard, PayPal, BankTransfer, ...` **kế thừa (inherits)** từ `PaymentMethod`.
- `PaymentProcessor` **chứa (has-a)** danh sách các đối tượng `PaymentMethod`.

2.2.3 Gợi ý cách làm

1. **Bước 1 – Thiết kế lớp `PaymentMethod`**
2. **Bước 2 – Viết các lớp con kế thừa** Mỗi lớp cài đặt riêng `validate()` và `process_payment()`. Ví dụ: `CreditCard` kiểm tra số thẻ, `PayPal` kiểm tra email.
3. **Bước 3 – Xây dựng lớp `PaymentProcessor`**

3. BÀI TẬP VỀ NHÀ

3.1 Bài 1 – Hệ thống Quản lý Sinh viên

Yêu cầu:

- Xây dựng hệ thống quản lý sinh viên, cho phép đăng ký môn học, nhập điểm và tính GPA.
- Tổ chức ít nhất ba lớp: `Student`, `Course`, `Enrollment`.

Chức năng yêu cầu:

1. Đăng ký và hủy đăng ký môn học.
2. Nhập điểm cho sinh viên.
3. Tính GPA của từng sinh viên.
4. Thống kê điểm trung bình theo môn học.
5. Tìm kiếm sinh viên theo tên hoặc mã số.

Phân tích hệ thống:

- **Student:** Lưu thông tin cá nhân của sinh viên, bao gồm mã số, họ tên, email, năm sinh. Có khả năng tính tuổi (`calculate_age()`) và hiển thị thông tin.

- **Course**: Đại diện cho một học phần với mã, tên môn học, số tín chỉ và số sinh viên tối đa. Cho phép thêm/xóa sinh viên và thống kê số lượng đã đăng ký.
- **Enrollment**: Liên kết sinh viên với môn học, lưu điểm giữa kỳ, cuối kỳ, bài tập. Tính điểm trung bình và xếp loại học lực.

Hướng dẫn chức năng:

1. **Đăng ký / Hủy đăng ký môn học:**
   - Viết phương thức `add_student(self, student: Student)` trong lớp `Course`.
   - Kiểm tra trùng sinh viên và giới hạn số lượng bằng `max_students`.
   - Trong lớp `Enrollment`, tạo hàm `enroll(student, course)` để thêm quan hệ.
2. **Nhập điểm cho sinh viên:**
   - Trong lớp `Enrollment`, lưu các điểm `midterm, final, assignment`.
   - Viết hàm `update_scores(midterm, final, assignment)` để cập nhật điểm.
3. **Tính GPA cho sinh viên:**
   - Trong lớp `Student`, viết `calculate_gpa()`: duyệt tất cả các Enrollment của sinh viên, tính trung bình theo trọng số tín chỉ.
   - $\text{GPA} = \frac{\sum(\text{điểm trung bình môn} \times \text{tín chỉ})}{\sum \text{tín chỉ}}$
4. **Thống kê điểm theo môn học:**
   - Trong lớp `Course`, viết hàm `get_average_score()` duyệt danh sách Enrollment liên quan.
5. **Tìm sinh viên theo tên / ID:**
   - Viết hàm `find_student(keyword)` trong lớp quản lý (hoặc danh sách tổng hợp).
   - Cho phép tìm kiếm không phân biệt hoa thường bằng `lower()`.

3.2 Bài 2 – Hệ thống Quản lý Ngân hàng

Yêu cầu:

- Thiết kế hệ thống ngân hàng hỗ trợ nhiều loại tài khoản và quản lý giao dịch.
- Tối thiểu gồm 3 lớp chính: `BankAccount`, `SavingsAccount`, `CheckingAccount`.

Chức năng yêu cầu:

1. Mở tài khoản mới.
2. Gửi và rút tiền.
3. Chuyển khoản giữa hai tài khoản.
4. Xem lịch sử giao dịch.
5. Tính và cộng lãi cho tài khoản tiết kiệm.

Phân tích hệ thống:

- **BankAccount (base class)**: Quản lý số tài khoản, chủ tài khoản và số dư. Cung cấp các phương thức `deposit()`, `withdraw()`, `get_balance()`.
- **SavingsAccount**: Kế thừa `BankAccount`, thêm thuộc tính `interest_rate`, phương thức tính và cộng lãi (`calculate_interest(), apply_interest()`), giới hạn số lần rút tiền.
- **CheckingAccount**: Kế thừa `BankAccount`, có hạn mức thấu chi (overdraft) và phí giao dịch.
- **Transaction**: Ghi nhận lịch sử giao dịch: loại giao dịch, số tiền, thời điểm, số dư sau giao dịch.

Hướng dẫn chức năng:

1. **Mở tài khoản mới:**

- Tạo đối tượng từ lớp `SavingsAccount` hoặc `CheckingAccount`.
- Sinh mã số tài khoản ngẫu nhiên (dùng `uuid` hoặc `random.randint`).

2. **Gửi / Rút tiền:**
   - Dùng `deposit(amount)` để cộng vào balance.
   - Trong `withdraw(amount)`, kiểm tra số dư:
     - Nếu là `CheckingAccount`, cho phép rút âm trong giới hạn overdraft.
     - Nếu là `SavingsAccount`, giới hạn số lần rút.
   - Ghi lịch sử giao dịch bằng lớp `Transaction`.
3. **Chuyển tiền giữa tài khoản:**
   - Viết hàm `transfer(from_account, to_account, amount)`.
   - Thực hiện `from.withdraw(amount)` và `to.deposit(amount)`.
   - Xử lý lỗi khi số dư không đủ hoặc tài khoản không tồn tại.
4. **Xem lịch sử giao dịch:**
   - Trong lớp `BankAccount`, lưu danh sách Transaction.
   - Viết hàm `show_history()` in từng dòng: thời gian, loại, số tiền, số dư sau giao dịch.
5. **Tính lãi suất:**
   - Trong lớp `SavingsAccount`, thêm `calculate_interest()` trả về số lãi.
   - Gọi `apply_interest()` để cộng lãi vào số dư.

3.3 Bài 3 – Hệ thống Cửa hàng Trực tuyến

Chức năng yêu cầu:

1. Thêm/xóa sản phẩm trong giỏ hàng.
2. Tính tổng tiền sau thuế và phí vận chuyển.
3. Áp dụng mã giảm giá hoặc điểm thưởng.
4. Tạo đơn hàng và cập nhật trạng thái.

Phân tích hệ thống:

- **Product (base class):** Thuộc tính chung: mã sản phẩm, tên, giá, tồn kho, loại. Các lớp con: `Electronics`, `Clothing`, `Book`, mỗi lớp thêm thuộc tính riêng.
- **Customer:** Lưu thông tin khách hàng và điểm tích lũy (`loyalty_points`).
- **ShoppingCart:** Quản lý danh sách sản phẩm đã chọn, tính tổng tiền, áp dụng giảm giá.
- **Coupon:** Giảm giá theo phần trăm hoặc giá trị cố định, có ngày hết hạn.
- **Order:** Ghi nhận thông tin đặt hàng, vận chuyển, phương thức thanh toán, trạng thái đơn hàng.

Hướng dẫn chức năng:

1. **Quản lý sản phẩm:**
   - Lớp `Product` chứa thông tin cơ bản.
   - Các lớp con (`Electronics`, `Clothing`, `Book`) override `display_info()` để thêm chi tiết riêng.
2. **Thêm / Xóa sản phẩm khỏi giỏ hàng:**
   - Trong lớp `ShoppingCart`, dùng list `items`.
   - Viết hàm `add_product(product, quantity)` và `remove_product(product_id)`.
3. **Tính tổng tiền và giảm giá:**
   - `calculate_total()` duyệt qua tất cả sản phẩm trong giỏ, cộng giá × số lượng.

- Nếu có `Coupon`, trừ đi phần giảm giá.

4. **Áp dụng điểm thưởng:**
   - Trong `Customer`, trừ điểm tích lũy từ `loyalty_points`.
   - Mỗi lần mua hàng, cộng điểm = 1% giá trị đơn hàng.
5. **Tạo đơn hàng:**
   - Lớp `Order` chứa thông tin: khách hàng, danh sách hàng, tổng tiền, trạng thái (Pending/Delivered).
   - Viết `confirm_order()` và `update_status()` để thay đổi trạng thái đơn.

3.4 Bài 4 – Hệ thống Mô phỏng Giao thông Thông minh

Tính năng yêu cầu:

1. Mô phỏng di chuyển phương tiện theo thời gian.
2. Áp dụng quy tắc giao thông (dừng đèn đỏ, giữ khoảng cách).
3. Phát hiện và tránh va chạm.
4. Thống kê: tốc độ trung bình, thời gian chờ, mức độ tắc nghẽn.

Phân tích hệ thống:

- **Vehicle (base class):** Các lớp con: `Car`, `Motorcycle`, `Bus`, `Truck`. Mỗi xe có vị trí, tốc độ, hướng di chuyển và kích thước.
- **TrafficLight:** Có các trạng thái `red, yellow, green` và bộ đếm thời gian cho mỗi pha.
- **Road:** Định nghĩa số làn đường, tốc độ giới hạn, và sức chứa phương tiện.
- **Intersection:** Mô phỏng ngã tư, quản lý nhiều đường và đèn tín hiệu.
- **TrafficSimulation:** Sinh ngẫu nhiên xe, cập nhật trạng thái, kiểm tra va chạm, thu thập thống kê.

Hướng dẫn chức năng:

1. **Khởi tạo mô phỏng:**
   - Tạo danh sách phương tiện (`Vehicle`) với thuộc tính vị trí, tốc độ.
   - Dùng `random.randint` để sinh xe ngẫu nhiên.
2. **Cập nhật chuyển động:**
   - Viết `move()` trong lớp `Vehicle` để cập nhật vị trí mỗi "tick" thời gian.
   - Kiểm tra giới hạn tốc độ theo `Road`.
3. **Điều khiển đèn giao thông:**
   - Trong `TrafficLight`, xây dựng `switch_state()` theo bộ đếm thời gian.
4. **Phát hiện va chạm:**
   - Trong `TrafficSimulation`, kiểm tra nếu hai xe có vị trí trùng hoặc quá gần nhau → báo va chạm.
5. **Thống kê kết quả:**
   - In ra số xe đang di chuyển, tốc độ trung bình, số va chạm, thời gian chờ.

3.5 Bài 5* – Game RPG Đơn Giản

Chức năng yêu cầu:

1. Cơ chế chiến đấu theo lượt (turn-based).
2. Sử dụng kỹ năng và vật phẩm trong trận.
3. Hệ thống lên cấp và tăng chỉ số.
4. Kiểm tra thắng/thua, in nhật ký trận đấu.

Phân tích hệ thống:

- **Character (base class):** Lưu thông tin nhân vật: tên, cấp độ, HP/MP, tấn công, phòng thủ. Các phương thức: `take_damage(), heal(), level_up(), is_alive()`.
- **Các lớp con:** `Warrior`, `Mage`, `Archer` — mỗi lớp có chỉ số và kỹ năng riêng.
- **Skill:** Đại diện cho kỹ năng hoặc phép thuật, có chi phí MP và lượng sát thương/hồi phục.
- **Item:** Gồm bình máu, bình năng lượng, và trang bị; có thể được dùng trong trận chiến.
- **Inventory:** Quản lý vật phẩm của nhân vật.

Hướng dẫn chức năng:

1. **Tạo nhân vật:**
   - Lớp `Character` có phương thức `attack(target)`, `take_damage()`, `heal()`.
   - Các lớp con (`Warrior`, `Mage`, `Archer`) override `attack()`.
2. **Cơ chế chiến đấu:**
   - Viết vòng lặp `while` cho trận chiến: mỗi lượt một nhân vật tấn công, bên kia phản công.
   - Khi `hp <= 0`, kết thúc trận.
3. **Sử dụng kỹ năng (Skill):**
   - Lớp `Skill` có `mp_cost` và `damage`.
   - Khi nhân vật dùng skill, trừ MP, gây sát thương cho đối thủ.
4. **Vật phẩm (Item) và Kho đồ (Inventory):**
   - `Inventory` chứa danh sách `Item`.
   - `use_item()` để hồi máu hoặc năng lượng.
5. **Lên cấp:**
   - Khi thắng trận, nhân vật gọi `level_up()` → tăng HP, attack, defense.

Nộp bài: Chạy code bài tập trên Colab hoặc các IDE (.py), thực hiện lại demo và làm các bài tập về nhà theo yêu cầu. Nếu thực hiện trên các IDE thì nộp kèm các file code kèm theo 1 pdf kết quả chạy. Đặt tên MSSV_HoVaTen_Lab4.pdf và nộp theo deadline trên Courses.