Thuật ngữ: Method và Sending Messages

Trong lập trình hướng đối tượng (OOP), có hai khái niệm quan trọng: **method** và **sending messages(gửi thông điệp)**.

- **Method (phương thức)**: là một hành vi được định nghĩa bên trong một đối tượng. Ví dụ trong lớp **BottlesOfBeer**, có một method tên là `song`.
- **Sending a message(gửi thông điệp)**: nghĩa là **yêu cầu một đối tượng thực hiện một hành vi cụ thể**. Trong ví dụ trước **BottlesOfBeer** , khi gọi `self.verses(...)` bên trong `song`, tức là `song` **gửi thông điệp "verses"** đến chính đối tượng `self`.

Nói ngắn gọn: **method là thứ bạn định nghĩa**, còn **message là thứ bạn gửi để kích hoạt hành vi**.

Tại sao lại dùng từ sending messages(gửi thông điệp) thay vì "gọi hàm"?

Trong nhiều ngôn ngữ lập trình, người ta hay dùng từ **"function"** thay cho "method", hoặc **"call"** thay vì "send". Nghe thì có vẻ giống nhau, nhưng thật ra hơi khác:

- Khi bạn nói **"gọi một hàm"**, nghe như bạn **đang điều khiển** hành vi đó, bạn biết chính xác nó sẽ làm gì.
- Nhưng nếu bạn nói **"gửi một thông điệp"**, nghĩa là **bạn chỉ đưa ra yêu cầu**, còn việc xử lý ra sao là chuyện của đối tượng.

Cách suy nghĩ này giúp **giảm sự phụ thuộc** giữa các phần trong chương trình. Người gửi thông điệp **không cần biết chi tiết bên trong người nhận xử lý ra sao**, và điều đó giúp cho code dễ bảo trì, dễ mở rộng hơn.

Trong cuốn sách này (và trong nhiều tài liệu về OOP), người ta thường dùng khái niệm **"gửi thông điệp"** thay vì **"gọi hàm"**, vì nó thể hiện đúng tinh thần của lập trình hướng đối tượng: **giao tiếp giữa các đối tượng một cách linh hoạt và độc lập**.

Ví dụ: Gọi một hàm – kiểm soát hành vi

```
def make_coffee():
    print("Boil water")
    print("Grind coffee beans")
    print("Brew coffee")
    print("Pour into cup")
    print("Done!")

make_coffee()
```

Ở đây, bạn **gọi hàm** `make_coffee()` và bạn **biết chắc từng bước bên trong nó là gì**: đun nước, xay cà phê, pha, rót ra ly...

Bạn kiểm soát hoàn toàn logic bên trong. Không có sự "ẩn ý", bạn **phụ thuộc vào chi tiết cụ thể** của hàm đó.

Ví dụ: Gửi thông điệp – yêu cầu hành vi, không quan tâm chi tiết

```
class CoffeeMachine:
    def make_coffee(self):
        self._heat_water()
        self._grind_beans()
        self._brew()
        self._pour()

    # Các method private bên trong
    def _heat_water(self): ...
    def _grind_beans(self): ...
    def _brew(self): ...
    def _pour(self): ...

machine = CoffeeMachine()
machine.make_coffee()
```

Lúc này, bạn **gửi thông điệp** `make_coffee` **đến** `machine`, yêu cầu nó pha cà phê. Nhưng **bạn không cần biết** nó sẽ đun nước hay pha như thế nào, chỉ cần biết nó **sẽ pha được cà phê** là đủ.

Ví dụ 2 "Gọi hàm" vs "Gửi thông điệp" trong bối cảnh phần mềm doanh nghiệp (ERP) – cụ thể là quy trình tạo đơn đặt hàng (Purchase Order).

Gọi hàm – kiểu procedural, kiểm soát mọi bước

Bạn viết một hàm và kiểm soát chi tiết quy trình tạo đơn hàng.

```
def create_purchase_order(supplier, items):
    # Kiểm tra tồn kho
    for item in items:
        if not check_inventory(item):
            raise Exception(f"{item} is not available in inventory")

    # Tính tổng tiền
    total = sum(item.price * item.quantity for item in items)

    # Tạo đơn hàng
    po = {
        "supplier": supplier,
        "items": items,
        "total": total,
    }

    # Gửi email xác nhận
    send_email(supplier.email, f"PO Created: Total = {total}")
    return po
```

Bạn kiểm soát hết mọi bước. Nếu muốn thay đổi logic, ví dụ thay đổi cách gửi email hay kiểm kho, bạn phải sửa trực tiếp trong hàm → **khó bảo trì, không theo tư duy hướng đối tượng**.

Gửi thông điệp – kiểu OOP, giao trách nhiệm cho đối tượng

```
class PurchaseOrder:
    def __init__(self, supplier, items):
        self.supplier = supplier
        self.items = items
        self.total = self.calculate_total()

    def calculate_total(self):
        return sum(item.price * item.quantity for item in self.items)

    def validate(self):
        for item in self.items:
            item.validate_inventory()

    def confirm(self):
        self.validate()
```

```
        self.send_confirmation_email()

    def send_confirmation_email(self):
        EmailService.send(to=self.supplier.email, subject="PO Confirmed", body=str(self))

# Cách sử dụng
po = PurchaseOrder(supplier, items)
po.confirm()  # <== GỬI THÔNG ĐIỆP "confirm"
```

Lợi ích:

- Gửi thông điệp `confirm()` mà không cần biết chi tiết nội bộ.
- Có thể thay `EmailService` bằng service khác mà không ảnh hưởng nơi gọi.
- Tách logic thành nhiều lớp – mỗi lớp có trách nhiệm rõ ràng.

Tóm lại:

- **Gọi hàm**: Bạn biết (và phải biết) nó làm gì – **bạn điều khiển chi tiết**.
- **Gửi thông điệp**: Bạn chỉ cần kết quả – **bạn tin rằng đối tượng sẽ làm đúng nhiệm vụ** , còn chi tiết xử lý thì để nó lo.

Cách “gửi thông điệp” chính là tư duy quan trọng của lập trình hướng đối tượng – tạo nên **sự độc lập, dễ thay đổi, và dễ bảo trì** trong phần mềm.

Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Phiên bản #3
Được tạo 19 tháng 4 2025 04:03:55 bởi Đỗ Ngọc Tú
Được cập nhật 19 tháng 4 2025 04:35:52 bởi Đỗ Ngọc Tú