Chương 7: Từ Module đến Đối tượng

Mục tiêu học tập

Sau khi học chương này, bạn sẽ có thể:

- Thiết kế các module và lớp với **tính kết hợp cao** và **mức độ liên kết thấp**.
- Hiểu được tầm quan trọng của **che giấu thông tin (information hiding)**.
- Mô tả các khía cạnh kỹ thuật phần mềm của **kế thừa, đa hình và liên kết động**.
- Phân biệt các khái niệm **tổng quát hóa (generalization), kết hợp (aggregation) và liên kết (association)**.
- Hiểu sâu hơn về lập trình hướng đối tượng và cách nó phát triển từ lập trình theo module.

7.1 Module là gì?

Trong các hệ thống phần mềm lớn, nếu toàn bộ mã nguồn chỉ là một khối lớn không phân tách, thì **bảo trì** sẽ cực kỳ khó khăn. Để giải quyết vấn đề này, chúng ta **chia nhỏ phần mềm** thành các phần nhỏ hơn, gọi là **module**.

Định nghĩa module

- **Stevens, Myers, và Constantine (1974):** Một module là một tập hợp các câu lệnh chương trình có **tên riêng**, có thể được gọi từ các phần khác của hệ thống.
- **Yourdon và Constantine (1979):** Một module là một **chuỗi câu lệnh liên tục**, được giới hạn bởi các **phần tử ranh giới** (như {...} trong C++ và Java).

Tại sao phải chia phần mềm thành module?

1. **Dễ bảo trì**: Khi có lỗi, có thể sửa một module mà không ảnh hưởng đến toàn bộ hệ thống.
2. **Dễ mở rộng**: Có thể thêm tính năng mới bằng cách tạo hoặc mở rộng module.
3. **Dễ đọc hiểu**: Chia nhỏ giúp mã nguồn dễ hiểu hơn.
4. **Tái sử dụng mã nguồn**: Một module có thể được sử dụng lại ở nhiều phần mềm khác.

7.2 Độ kết hợp (Cohesion)

Là mức độ liên quan giữa các thành phần **bên trong một module**.

7 cấp độ độ kết hợp (từ thấp đến cao)

1. **Kết hợp ngẫu nhiên (Coincidental Cohesion) – Xấu nhất**
    - Module thực hiện nhiều tác vụ **không liên quan**.

- Ví dụ: Một module vừa in dữ liệu, vừa đảo chuỗi, vừa thực hiện phép tính số học.
- Giải pháp: **Tách nhỏ module thành các module chuyên biệt**.

2. **Kết hợp logic (Logical Cohesion)**
   - Module thực hiện một nhóm thao tác **có liên quan**, nhưng thao tác cụ thể sẽ được **chọn bởi một tham số**.
   - Ví dụ: Một module có thể đọc dữ liệu từ bàn phím, từ file, hoặc từ cơ sở dữ liệu, tùy vào tham số truyền vào.
3. **Kết hợp theo thời gian (Temporal Cohesion)**
   - Module thực hiện **nhiều tác vụ xảy ra cùng một thời điểm**.
   - Ví dụ: Một module init_system() thực hiện khởi tạo file log, tải dữ liệu, mở kết nối mạng – nhưng các tác vụ này không liên quan trực tiếp.
4. **Kết hợp theo quy trình (Procedural Cohesion)**
   - Các thao tác **phải thực hiện theo trình tự nhất định**.
   - Ví dụ: Một module process_transaction() gồm các bước **đọc dữ liệu → xác thực → lưu vào cơ sở dữ liệu**.
5. **Kết hợp theo giao tiếp (Communicational Cohesion)**
   - Các thao tác sử dụng chung **cùng một dữ liệu**.
   - Ví dụ: Một module vừa **cập nhật dữ liệu vào cơ sở dữ liệu** vừa **ghi log thay đổi đó**.
6. **Kết hợp chức năng (Functional Cohesion)**
   - Module **chỉ thực hiện một nhiệm vụ duy nhất**.
   - Ví dụ: calculate_tax() chỉ thực hiện tính thuế.
7. **Kết hợp thông tin (Informational Cohesion) – Tốt nhất**
   - Một module có nhiều hàm độc lập, **nhưng tất cả đều làm việc trên cùng một cấu trúc dữ liệu**.
   - Ví dụ: Một class Employee có các hàm getSalary(), setSalary(), printEmployeeDetails() – tất cả đều làm việc với dữ liệu nhân viên.

Nguyên tắc thiết kế tốt: **Càng lên cao trong bảng trên, phần mềm càng dễ bảo trì và tái sử dụng.**

7.3 Độ liên kết (Coupling)

Là mức độ **phụ thuộc** giữa các module.

5 mức độ liên kết (từ cao đến thấp)

1. **Liên kết nội dung (Content Coupling) – Xấu nhất**
    - Một module thay đổi hoặc truy cập **trực tiếp** vào nội dung của module khác.
    - Ví dụ: Module A sửa đổi biến nội bộ của module B.
2. **Liên kết chung (Common Coupling)**
    - Hai module cùng truy cập vào **biến toàn cục**.
    - Ví dụ: Cả Module_A và Module_B đều có thể thay đổi global_variable.
3. **Liên kết điều khiển (Control Coupling)**
    - Một module truyền **tham số điều khiển** đến module khác để quyết định nó sẽ làm gì.
    - Ví dụ: process_data(mode, data) với mode quyết định cách xử lý.
4. **Liên kết con dấu (Stamp Coupling)**
    - Một module truyền **cả một cấu trúc dữ liệu lớn** nhưng chỉ sử dụng một phần.
    - Ví dụ: calculate_salary(employee_record) chỉ dùng lương nhưng truyền toàn bộ hồ sơ nhân viên.
5. **Liên kết dữ liệu (Data Coupling) – Tốt nhất**
    - Module chỉ truyền dữ liệu **thật sự cần thiết**.
    - Ví dụ: calculate_tax(salary) chỉ nhận lương.

Nguyên tắc thiết kế tốt: **Giảm mức độ liên kết xuống càng thấp càng tốt**.

7.4 Đóng gói dữ liệu (Data Encapsulation)

- Là việc **kết hợp dữ liệu và các thao tác** trên dữ liệu đó vào cùng một đơn vị (ví dụ: Class trong OOP).
- Giúp:
    - **Che giấu chi tiết cài đặt**.
    - **Giảm tác động của thay đổi**.
    - **Bảo vệ dữ liệu** khỏi truy cập trái phép.

Ví dụ: Quản lý hàng đợi công việc (Job Queue)

- **Thiết kế cũ:** Các module riêng lẻ trực tiếp truy cập vào hàng đợi → **khó bảo trì**.
- **Thiết kế mới:** Đóng gói hàng đợi vào một class JobQueueClass với các hàm addJob(), removeJob() → **dễ thay đổi cách triển khai** mà không ảnh hưởng đến code khác.

7.5 Kiểu dữ liệu trừu tượng (Abstract Data Type - ADT)

- Là **một kiểu dữ liệu** đi kèm với **các phép toán trên nó**.
- Ví dụ: **Class JobQueueClass** trong lập trình hướng đối tượng (OOP) chính là một ADT.
- ADT giúp phần mềm:
    - **Độc lập với cách triển khai**.
    - **Dễ tái sử dụng**.
    - **Bảo mật dữ liệu** tốt hơn.