Câu 1

SOLID là tập hợp 5 nguyên lý thiết kế hướng đối tượng giúp phần mềm dễ bảo trì, dễ mở rộng và hạn chế lỗi. Bao gồm:

- S – Single Responsibility Principle (Nguyên lý đơn trách nhiệm)

+ Định nghĩa: Mỗi lớp chỉ nên có một lý do để thay đổi, tức là chỉ đảm nhận một trách nhiệm duy nhất.

+ Ý nghĩa: Giúp mã nguồn dễ đọc, dễ test, giảm sự phụ thuộc giữa các chức năng.

+ Ví dụ: Một lớp InvoicePrinter chỉ đảm nhận việc in hóa đơn, không kiêm cả việc tính toán tổng tiền.

- O – Open/Closed Principle (Nguyên lý đóng/mở)

+ Định nghĩa: Phần mềm nên mở rộng được mà không cần sửa đổi mã nguồn cũ.

+ Ý nghĩa: Khi thêm chức năng mới, ta không phải thay đổi lớp cũ mà chỉ cần mở rộng (inheritance, interface, strategy pattern…).

+ Ví dụ: Một lớp Shape có phương thức draw(). Khi thêm hình Triangle, ta chỉ cần tạo lớp mới Triangle implements Shape, không cần sửa lớp Shape.

- L – Liskov Substitution Principle (Nguyên lý thay thế Liskov)

+ Định nghĩa: Các lớp con có thể thay thế lớp cha mà không làm thay đổi tính đúng đắn của chương trình.

+ Ý nghĩa: Đảm bảo tính kế thừa đúng nghĩa. Nếu S là lớp con của T, thì có thể dùng S ở bất kỳ nơi nào chấp nhận T.

+ Ví dụ sai: Lớp Penguin kế thừa Bird nhưng Bird có phương thức fly(). Penguin không bay được nên vi phạm nguyên lý này.

- I – Interface Segregation Principle (Nguyên lý phân tách interface)

+ Định nghĩa: Một lớp không nên bị ép cài đặt các phương thức mà nó không cần dùng.

+ Ý nghĩa: Nên tách các interface lớn thành nhiều interface nhỏ, đặc thù hơn.

+ Ví dụ: Thay vì một interface IMachine có cả print(), scan(), fax(), ta nên chia thành IPrinter, IScanner, IFax.

- D – Dependency Inversion Principle (Nguyên lý đảo ngược phụ thuộc)

+ Định nghĩa: Các module cấp cao không nên phụ thuộc trực tiếp vào module cấp thấp, mà cả hai nên phụ thuộc vào abstraction (interface).

+ Ý nghĩa: Giảm sự phụ thuộc cứng giữa các lớp, dễ thay thế hoặc mở rộng.

+ Ví dụ: Thay vì OrderService dùng trực tiếp MySQLDatabase, nó nên phụ thuộc vào IDatabase. Khi đổi sang MongoDBDatabase, chỉ cần thay đổi phần implement.

Câu 2: DI và IoC

IoC (Inversion of Control)

+ Khái niệm: Là nguyên tắc trong đó control flow (dòng điều khiển, việc tạo đối tượng, quản lý vòng đời, gọi phương thức…) không do lập trình viên trực tiếp điều khiển, mà do framework/container đảm nhiệm.

+ Ý nghĩa: Giúp code ít phụ thuộc hơn, dễ mở rộng và dễ test.

+ Ví dụ: Trong Spring Framework, ta không cần new Service(), mà Spring IoC container sẽ tự khởi tạo và inject vào khi cần.

DI (Dependency Injection)

- Khái niệm: Là cách cụ thể để thực hiện IoC, tức là đưa (inject) các dependency vào lớp thay vì lớp tự tạo ra.

- Các hình thức DI:

+ Constructor Injection (truyền dependency qua constructor).

+ Setter Injection (truyền dependency qua setter).

+ Field Injection (gán trực tiếp vào thuộc tính – thường kết hợp với annotation như @Autowired trong Spring).