|  |  |
| --- | --- |
| **1. SOLID là gì ?** | -SOLID là viết tắt của các từ S(single-responsiblity principle: nguyên lý trách nhiệm  duy nhất O(Open-closed principle): nguyên lý đóng mở L(Liskov substitution principle): nguyên lý thay thế  liskov I(Interface segregation principle): nguyên lý phân tách  interface D(Dependency Invertion Principle):nguyên lý đảo ngược phụ thuộc -SOLID bao gồm các nguyên lý quan trọng bật nhất cần  tuân thủ khi thiết kế kiến trúc phần mềm nhằm đạt  được mục đích: + dể mở rộng + dể bảo trì |
| **2. Ý nghĩa của từng chữ cái ? cho ví dụ** | **- S - Single Responsibility Principle (Nguyên tắc đơn trách nhiệm) :**  + Một lớp chỉ nên có một lý do để thay đổi, tức là nó chỉ chịu trách nhiệm về một chức năng cụ thể.  Ví dụ :  class ReportGenerator {  void generateReport() {  // Code tạo báo cáo  }  }  - **O - Open/Closed principle(Nguyên tắc mở/đóng) :**  + Lớp nên mở rộng được mà không cần chỉnh sửa trực tiếp  Ví dụ :  abstract class Shape {  abstract double getArea();  }  class Circle extends Shape {  double radius;  Circle(double radius) { this.radius = radius; }  @Override  double getArea() {  return Math.PI \* radius \* radius;  }  }  - **L - Liskov Substitution princile (Nguyên tắc thay thế Liskov) :**  - Lớp con phải có thể thay thế lớp cha mà không làm thay đổi hành vi mong đợi.  Ví dụ :  class Rectangle {  int width, height;  void setWidth(int width) { this.width = width; }  void setHeight(int height) { this.height = height; }  }  class Square extends Rectangle {  void setWidth(int width) { super.width = super.height = width; }  void setHeight(int height) { super.width = super.height = height; }  }  - **I - Interface Segregation Principle (Nguyên tắc phân chia giao diện)**  + Không nên ép các lớp phải phụ thuộc vào những phương thức mà chúng không cần.  Ví dụ :  interface Worker {  void work();  }  interface Eater {  void eat();  }  class Human implements Worker, Eater {  public void work() { System.out.println("Working..."); }  public void eat() { System.out.println("Eating..."); }  }  - **D - Dependency Inversion Principle (Nguyên tắc đảo ngược phụ thuộc)**  **+** Các module cấp cao không nên phụ thuộc trực tiếp vào module cấp thấp, mà nên phụ thuộc vào abstraction.  Ví dụ :  interface MessageService {  void sendMessage(String message);  }  class EmailService implements MessageService {  public void sendMessage(String message) {  System.out.println("Email: " + message);  }  }  class Notification {  private MessageService service;  Notification(MessageService service) { this.service = service; }  void notifyUser() {  service.sendMessage("Hello User!");  }  } |