# **Lab 3: Use AI for Static Code Analysis**

**Họ và tên:** Bùi Mạnh Thịnh

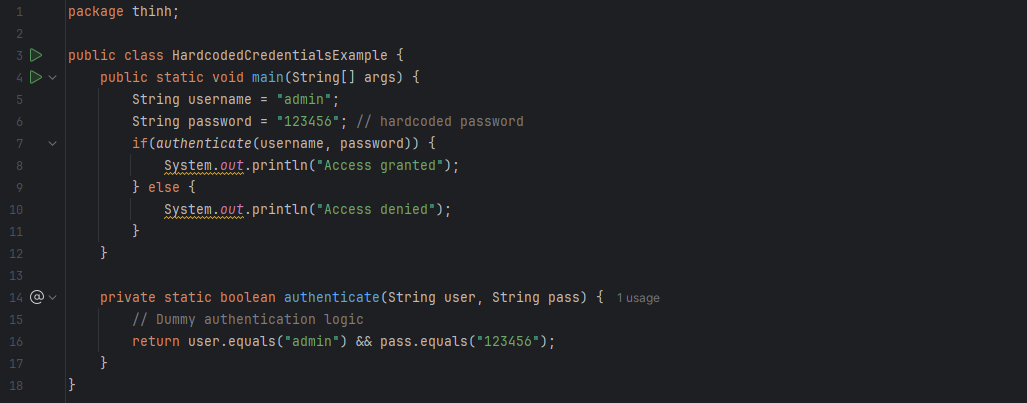
## **Phân tích các lỗi thường gặp trong lập trình**

## Một số lỗi hay gặp khi viết chương trình:

* Code smells
* Null pointer
* SQL injection
* Dead code
* Off-by-one error
* Hardcoded credentials

## **Bài tập 1: HardcodedCredentialsExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Hardcoded credentials**

* Thông tin đăng nhập (username, password) được viết cứng trực tiếp trong mã nguồn
* Loại lỗi: **Security vulnerability** (Lỗ hổng bảo mật)

**Lỗi 2: Sử dụng trực tiếp System.out.println để log**

* Loại lỗi: **Code smell** và **Bad Practice** (Thực hành lập trình không tốt)

### **Giải thích tại sao đây là lỗi**

**Hardcoded credentials** là lỗi nghiêm trọng vì:

* Thông tin đăng nhập khi được viết trực tiếp trong mã nguồn rất dễ bị lộ nếu mã nguồn bị chia sẻ, bị leak hoặc bị tấn công
* Việc thay đổi thông tin đăng nhập sẽ rất khó khăn vì phải sửa trực tiếp trong mã nguồn, dễ gây sai sót và làm giảm tính linh hoạt của chương trình
* Đây là vi phạm nguyên tắc bảo mật cơ bản trong lập trình

**Sử dụng trực tiếp System.out.println** để log là một thực hành không tốt vì:

* Không hỗ trợ phân loại log theo mức độ như INFO, DEBUG, ERROR
* Không cho phép ghi log ra file để lưu trữ và kiểm tra lịch sử hoạt động
* Không thể cấu hình hoặc tắt log trong môi trường production
* Việc log thủ công như vậy làm giảm khả năng kiểm soát và quản lý log khi ứng dụng phức tạp hơn

### **Đề xuất cách sửa**

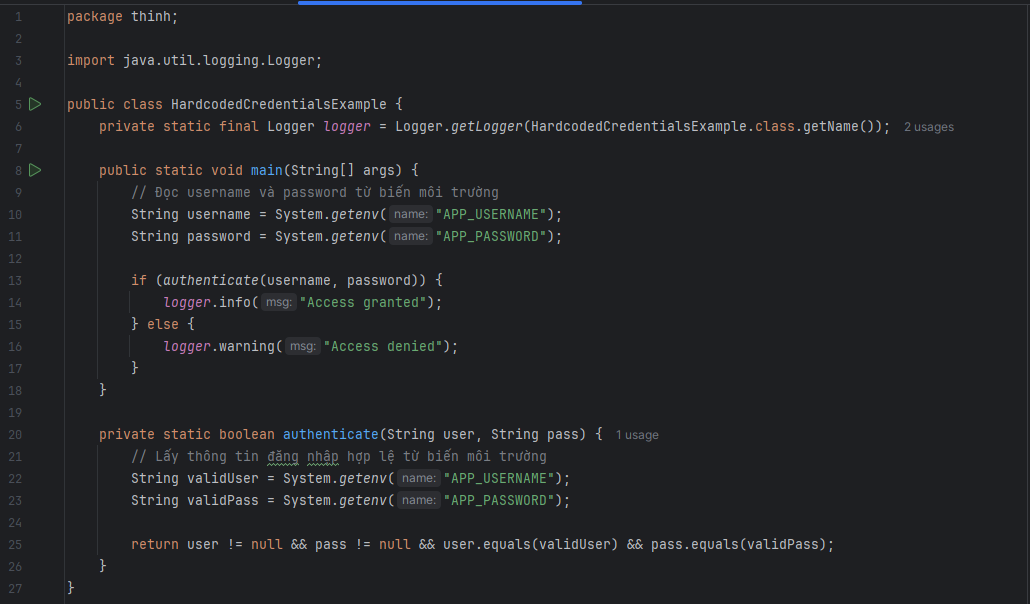
**Đối với lỗi Hardcoded credentials:**

* Thay thế cách khai báo tài khoản và mật khẩu trực tiếp trong mã nguồn bằng cách:
  + Đọc từ biến môi trường (Environment Variables)
  + Hoặc đọc từ các file cấu hình như .properties, .env để đảm bảo bảo mật tốt hơn và dễ dàng bảo trì, cập nhật thông tin đăng nhập

**Đối với lỗi sử dụng trực tiếp System.out.println:**

* Thay thế bằng việc sử dụng các thư viện logging chuyên nghiệp như:
  + java.util.logging cho các ứng dụng cơ bản
  + Log4j hoặc SLF4J kết hợp với Logback cho các ứng dụng thực tế hoặc dự án lớn
* Việc sử dụng các thư viện này cho phép phân loại log, ghi log ra file, hỗ trợ cấu hình và dễ dàng quản lý log trong môi trường phát triển và môi trường production

### **Viết lại code sạch hơn**



## **Bài tập 2: PathTraversalExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Path Traversal**

* Loại lỗi: **Security vulnerability** (Lỗ hổng bảo mật)
* Chương trình cho phép người dùng nhập vào đường dẫn tập tin một cách không kiểm soát, có thể bị tấn công để truy cập các tập tin nhạy cảm bên ngoài thư mục cho phép

**Lỗi 2: Sử dụng trực tiếp System.out.println để log**

* Loại lỗi: **Code smell** – Thực hành không tốt

### **Giải thích tại sao đây là lỗi**

**Lỗi 1: Path Traversal**

* Đây là một lỗi bảo mật nghiêm trọng vì ứng dụng cho phép người dùng nhập đường dẫn tập tin và mở tập tin đó mà không kiểm tra hợp lệ
* Nếu người dùng nhập giá trị như "../../../../etc/passwd" hoặc "../secret.txt" (như ví dụ), họ có thể truy cập được các tập tin quan trọng của hệ thống hoặc tập tin bí mật bên ngoài thư mục an toàn
* Lỗi này thường bị khai thác để đọc các tập tin cấu hình, mật khẩu, hoặc tập tin hệ thống

**Lỗi 2: Sử dụng trực tiếp System.out.println**

* Việc sử dụng System.out.println để log là cách ghi log đơn giản nhưng không chuyên nghiệp
* Không hỗ trợ phân loại log theo mức độ (INFO, DEBUG, ERROR)
* Không ghi log ra file, không quản lý được log khi ứng dụng phức tạp

### **Đề xuất cách sửa**

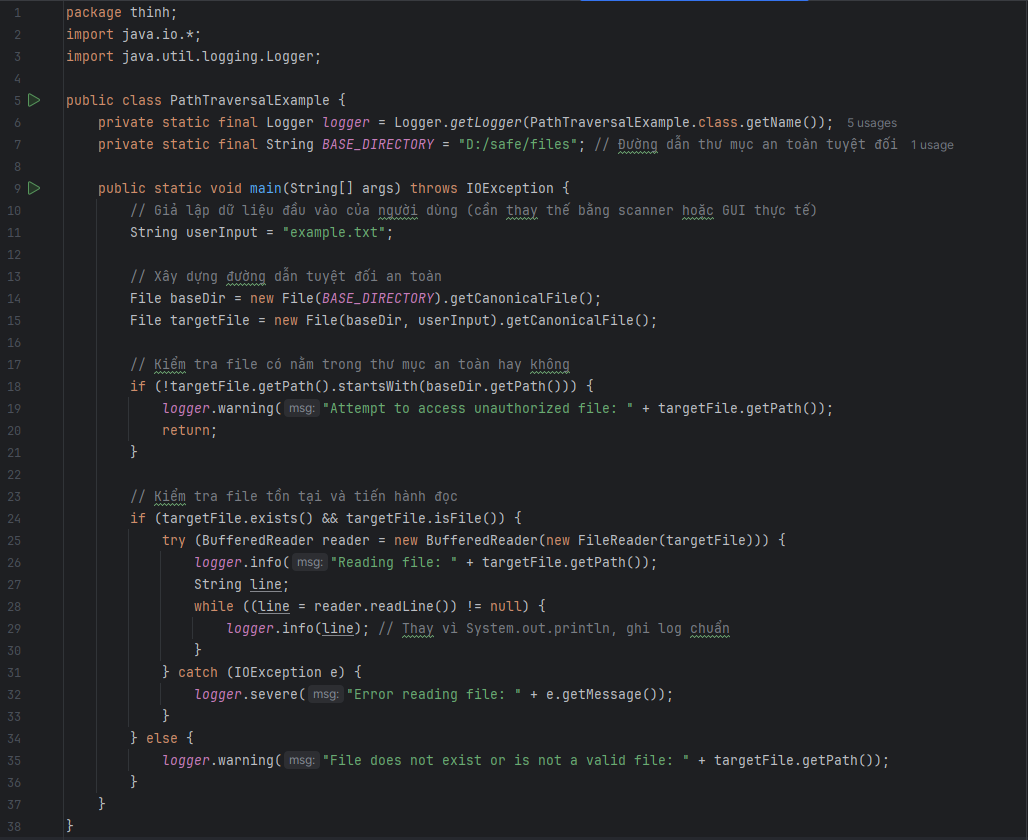
**Lỗi 1: Path Traversal**

* Cần kiểm tra và giới hạn đường dẫn mà người dùng nhập vào
* Chỉ cho phép người dùng truy cập trong thư mục được phép (thư mục an toàn)
* Nên sử dụng các phương pháp kiểm tra như:
  + Chuẩn hóa đường dẫn để loại bỏ .. (normalize path)
  + Kiểm tra xem đường dẫn tuyệt đối có nằm trong thư mục cho phép hay không
  + Có thể chỉ cho phép người dùng chọn tập tin từ danh sách đã định sẵn thay vì nhập đường dẫn tùy ý

**Lỗi 2: Sử dụng trực tiếp System.out.println**

* Nên sử dụng thư viện logging như java.util.logging, Log4j hoặc SLF4J để quản lý log hiệu quả hơn
* Logging nên được phân cấp mức độ như INFO, WARNING, ERROR để dễ theo dõi

### **Viết lại code sạch hơn**



## **Bài tập 3: ResourceLeakExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Resource Leak**

* Loại lỗi: **Runtime Error** (Rò rỉ tài nguyên)

**Lỗi 2: Sử dụng trực tiếp System.out.println để log**

* Loại lỗi: **Code Smell**

**Lỗi 3: Sử dụng e.printStackTrace()**

* Loại lỗi: **Logging không chuyên nghiệp**

### **Giải thích tại sao đây là lỗi**

**Lỗi 1: Resource Leak**

* Khi sử dụng BufferedReader, nếu không đóng (close) đối tượng sau khi sử dụng, sẽ dẫn đến rò rỉ tài nguyên file
* Trường hợp này nếu exception xảy ra, file sẽ không được đóng, gây nguy cơ chiếm tài nguyên lâu dài

**Lỗi 2: Sử dụng trực tiếp System.out.println**

* Đây là thói quen không tốt trong việc quản lý log
* Không thể phân loại log, không thể ghi log ra file, không thể quản lý log tập trung

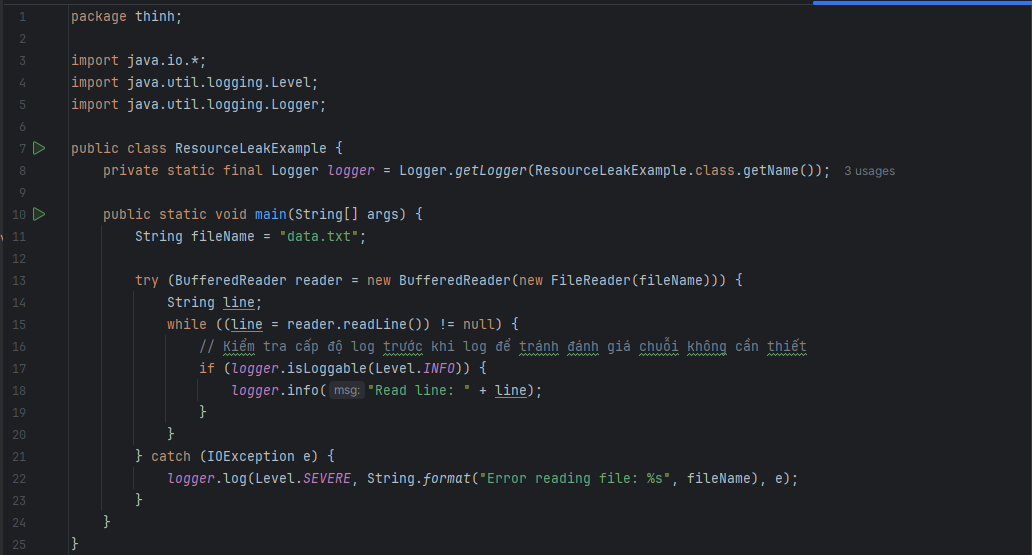
**Lỗi 3: Sử dụng e.printStackTrace()**

* Đây là cách xử lý lỗi thiếu chuyên nghiệp, chỉ in stack trace ra console mà không quản lý được log
* Cũng gây ra vấn đề tương tự như lỗi thứ 2

### **Đề xuất cách sửa**

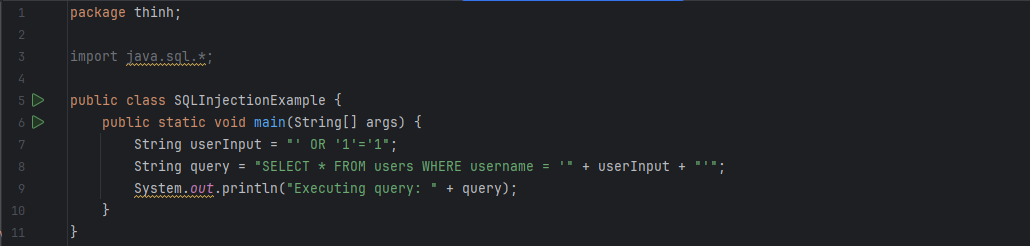
* Sử dụng **try-with-resources** để tự động đóng BufferedReader và tránh rò rỉ tài nguyên
* Sử dụng thư viện logging như java.util.logging để thay thế hoàn toàn System.out.println và e.printStackTrace()
* Ghi log theo cấp độ (INFO, WARNING, SEVERE)

### **Viết lại code sạch hơn**



## **Bài tập 4: SQLInjectionExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: SQL Injection**

* Loại lỗi: **Security vulnerability** (Lỗ hổng bảo mật nghiêm trọng)

**Lỗi 2: Truy vấn lấy tất cả các cột**

* Loại lỗi: **Performance Issue** (SQL queries should retrieve only necessary fields)

**Lỗi 3: Hardcoded log**

* Loại lỗi: **Code Smell** (Sử dụng trực tiếp System.out.println để log)

**Lỗi 4: Logging argument được đánh giá ngay cả khi log không được bật**

* Loại lỗi: **Performance Issue** (Preconditions and logging arguments should not require evaluation)

### **Giải thích tại sao đây là lỗi**

**Lỗi 1: SQL Injection**

Nối trực tiếp chuỗi đầu vào của người dùng vào câu lệnh SQL:  
 String query = "SELECT \* FROM users WHERE username = '" + userInput + "'";

* Đây là lỗ hổng SQL Injection, có thể bị kẻ tấn công lợi dụng để truy xuất hoặc thao túng dữ liệu trái phép

**Lỗi 2: Truy vấn tất cả các cột**

* Việc dùng SELECT \* làm giảm hiệu năng và có nguy cơ rò rỉ dữ liệu nhạy cảm nếu bảng có nhiều cột không cần thiết

**Lỗi 3: Sử dụng System.out.println để log**

* Việc ghi log bằng System.out.println không chuyên nghiệp, không có mức độ log, không quản lý được log trong hệ thống lớn

**Lỗi 4: Logging argument luôn được đánh giá**

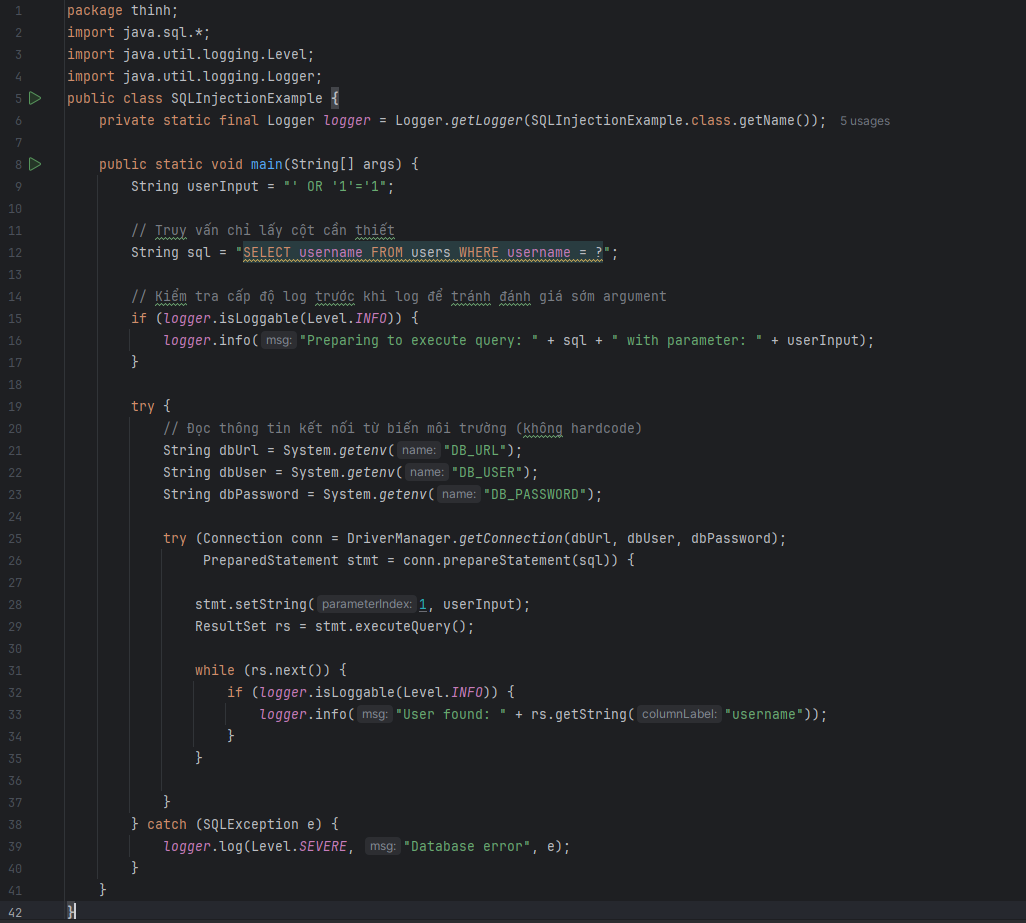
Khi viết:  
 System.out.println("Executing query: " + query);

* Biểu thức "Executing query: " + query luôn được đánh giá dù log có được ghi hay không
* Trong các hệ thống lớn, việc log nhiều câu lệnh như vậy gây lãng phí tài nguyên nếu không kiểm soát được cấp độ log

### **Đề xuất cách sửa**

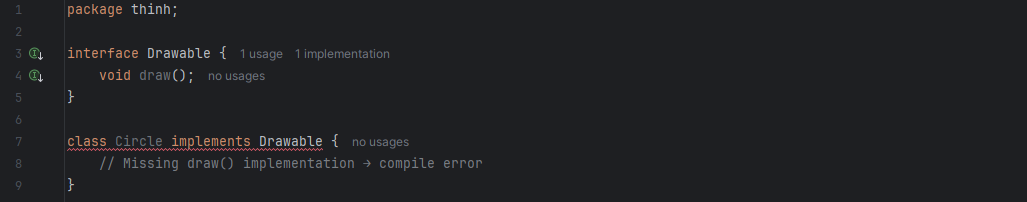
* Sử dụng **PreparedStatement** để ngăn SQL Injection
* Chỉ truy vấn những cột cần thiết (ví dụ: username)
* Thay System.out.println bằng logging framework như java.util.logging hoặc SLF4J
* Đảm bảo log được đánh giá lười (lazy evaluation) hoặc kiểm tra cấp độ log nếu cần

### **Viết lại code sạch hơn**



## **Bài tập 5: UnimplementedInterfaceExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi: Vi phạm nguyên tắc giao diện (Interface Contract Violation)**

* Loại lỗi: **Compile Error**

### **Giải thích tại sao đây là lỗi**

* Interface Drawable yêu cầu tất cả các lớp cài đặt phải triển khai phương thức draw()
* Trong lớp Circle, phương thức draw() chưa được viết
* **Kết quả:** chương trình sẽ không biên dịch được vì vi phạm hợp đồng interface

### **Đề xuất cách sửa**

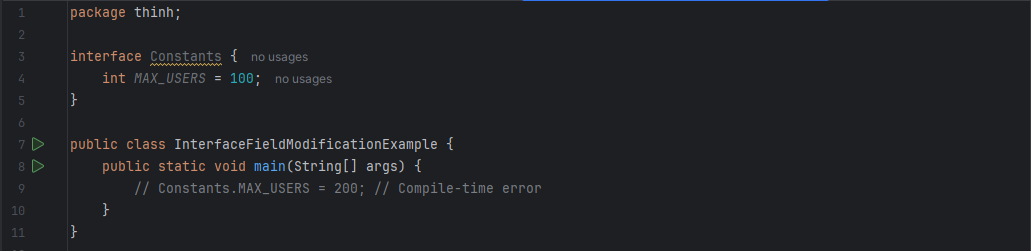
* Cần triển khai đầy đủ phương thức draw() trong lớp Circle để đáp ứng đúng yêu cầu của interface

### **Viết lại code sạch hơn**



## **Bài tập 6: InterfaceFieldModificationExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi: Vi phạm quy tắc bất biến của hằng số trong interface**

* Loại lỗi: **Compile-time error**

### **Giải thích tại sao đây là lỗi**

Các trường trong interface mặc định là public static final. Điều đó có nghĩa là:  
 int MAX\_USERS = 100;

thực chất tương đương với:  
 public static final int MAX\_USERS = 100;

* Hằng số final không thể bị gán lại giá trị mới

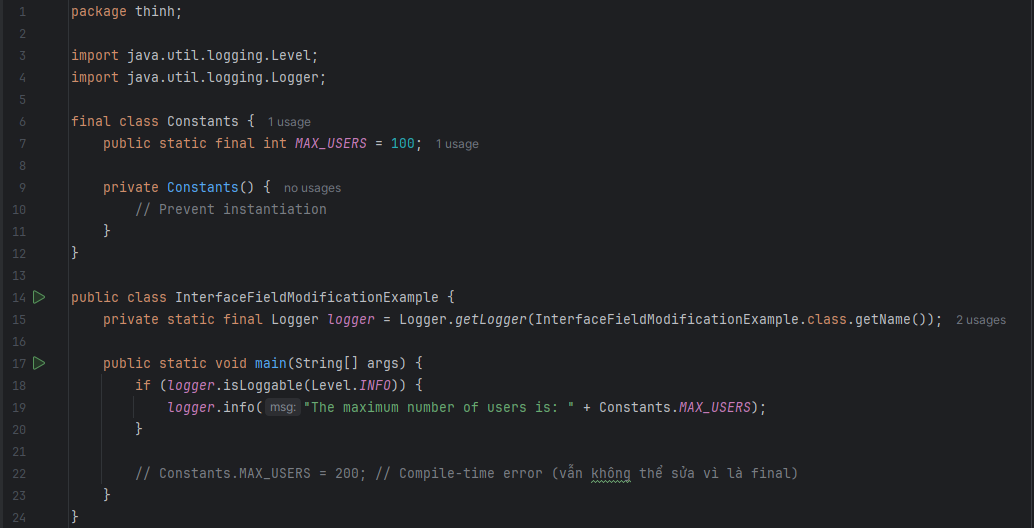
Việc cố gắng gán:  
 Constants.MAX\_USERS = 200;

* sẽ dẫn đến lỗi biên dịch vì vi phạm tính bất biến của hằng số

### **Đề xuất cách sửa**

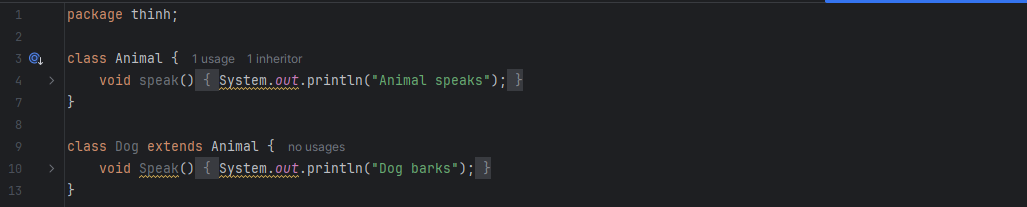
* Không được phép gán lại giá trị cho hằng số được khai báo trong interface
* Nếu cần thay đổi giá trị, phải thay đổi ngay tại nơi khai báo, hoặc nếu cần giá trị có thể thay đổi, hãy sử dụng biến trong class (non-final) thay vì interface

### **Viết lại code sạch hơn**



## **Bài tập 7: MissingOverrideAnnotationExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Method name inconsistency**

* Loại lỗi: **Logic Error** (Lỗi đặt tên phương thức không nhất quán)

**Lỗi 2: Vi phạm quy tắc ghi đè (Overriding)**

* Loại lỗi: **OOP Violation** trong lập trình hướng đối tượng

**Lỗi 3: Sử dụng trực tiếp System.out.println để log**

* Loại lỗi: **Code Smell**

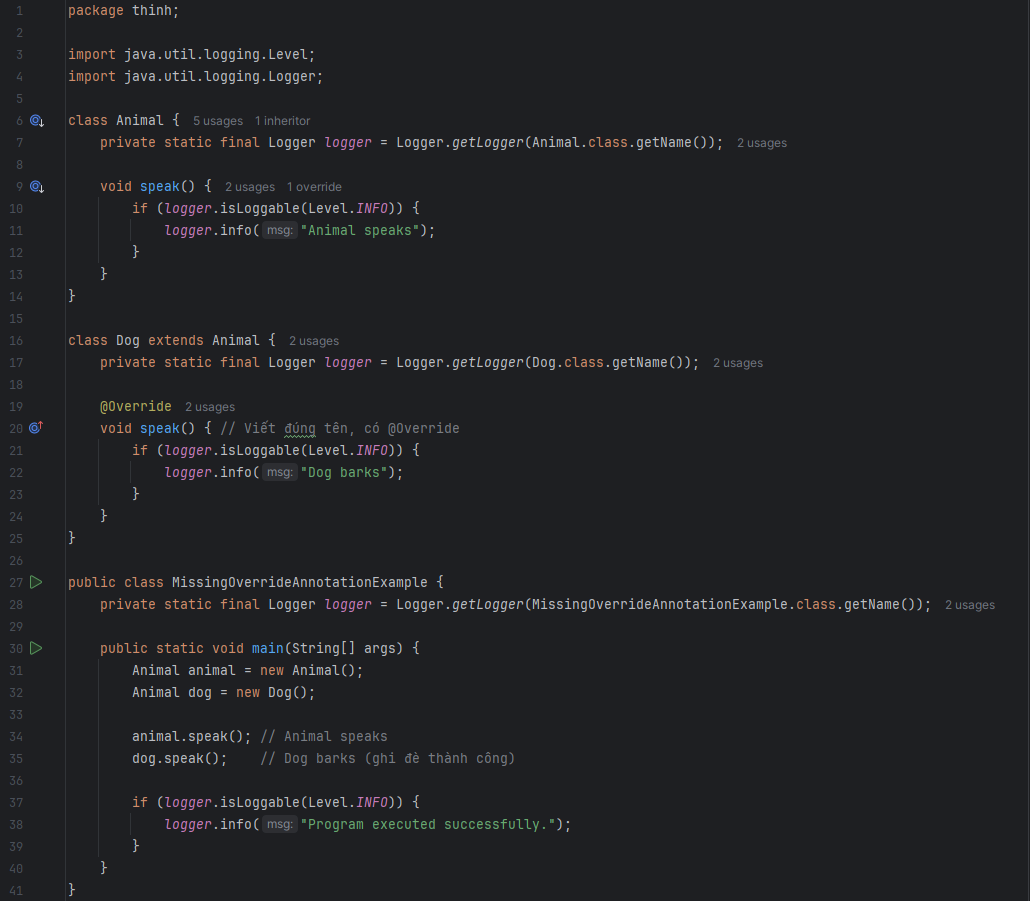
### **Giải thích tại sao đây là lỗi**

* Trong lớp Animal có phương thức speak(), nhưng trong lớp Dog lại viết là Speak() (khác nhau về chữ hoa)
* Java phân biệt chữ hoa chữ thường, vì vậy phương thức Speak() trong lớp Dog không ghi đè speak() của lớp Animal
* **Hậu quả:** Khi gọi speak() qua đối tượng Dog nhưng được tham chiếu bởi kiểu Animal, chương trình sẽ không gọi được phương thức mong muốn của Dog → gây hiểu nhầm và lỗi logic
* Sử dụng trực tiếp System.out.println để log cũng là code smell, cần thay bằng Logger

### **Đề xuất cách sửa**

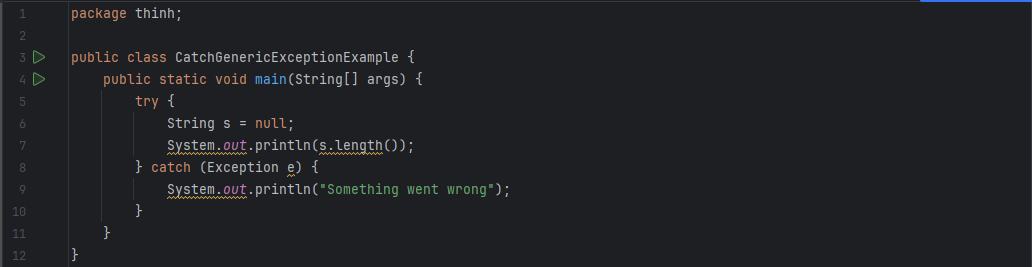
* Cần ghi đè phương thức đúng tên và có annotation @Override để dễ phát hiện lỗi nếu viết sai
* Sử dụng java.util.logging.Logger để log thay vì System.out.println

### **Viết lại code sạch hơn**



## **Bài tập 8: CatchGenericExceptionExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Bắt ngoại lệ quá tổng quát**

* Loại lỗi: **Exception Handling Issue** (Catching generic exception)

**Lỗi 2: Sử dụng trực tiếp System.out.println để log**

* Loại lỗi: **Code Smell** (Logging không chuẩn)

### **Giải thích tại sao đây là lỗi**

**Việc bắt ngoại lệ chung chung với catch (Exception e) sẽ:**

* Che giấu các loại lỗi cụ thể
* Gây khó khăn cho việc xử lý chính xác các tình huống lỗi
* Có thể làm bỏ sót các lỗi nghiêm trọng như OutOfMemoryError, StackOverflowError (mặc dù là unchecked exception nhưng bị bắt quá rộng cũng là anti-pattern)
* Khi bắt ngoại lệ, nên chỉ bắt loại ngoại lệ cụ thể mà chương trình có thể dự đoán (ví dụ: NullPointerException trong trường hợp này)

**Sử dụng System.out.println để log:**

* Sẽ không kiểm soát được mức độ log và không thể ghi ra file log khi cần

### **Đề xuất cách sửa**

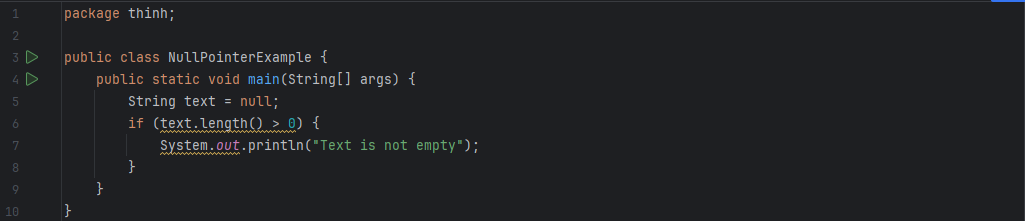
* Nên bắt ngoại lệ cụ thể, ví dụ NullPointerException
* Thay System.out.println bằng java.util.logging.Logger để quản lý log chuyên nghiệp hơn
* Có thể thêm thông tin chi tiết của ngoại lệ vào log

### **Viết lại code sạch hơn**



## **Bài tập 9: NullPointerExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Null pointers should not be dereferenced**

* Loại lỗi: **Runtime Error**

**Lỗi 2: Standard outputs should not be used directly to log anything**

* Loại lỗi: **Code Smell**

**Lỗi 3: Condition 'text != null' is missing**

* Loại lỗi: **Logic Error** (điều kiện cần kiểm tra null không được viết)

### **Giải thích tại sao đây là lỗi**

**Null pointer dereference:**

* Biến text được gán bằng null, nhưng ngay sau đó bạn gọi text.length() → điều này sẽ gây ra NullPointerException tại thời điểm runtime

**Thiếu kiểm tra null:**

* Trước khi truy cập bất kỳ phương thức hoặc thuộc tính nào của một đối tượng, bạn phải kiểm tra xem đối tượng đó có null hay không

**Sử dụng System.out.println để log:**

* Việc sử dụng trực tiếp System.out.println để log không phải là phương pháp chuẩn trong các ứng dụng thực tế. Bạn nên sử dụng Logger để quản lý và theo dõi log

### **Đề xuất cách sửa**

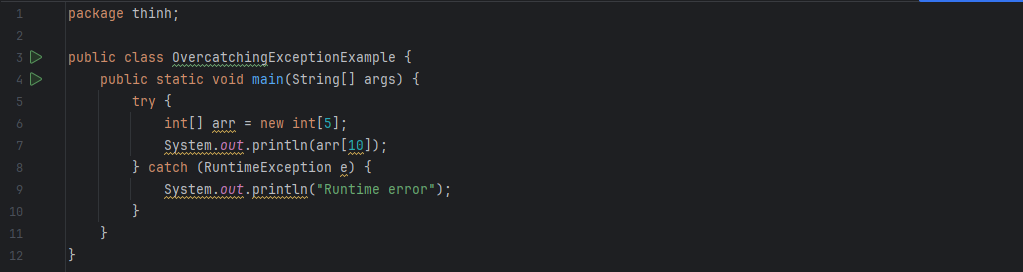
* Thêm kiểm tra điều kiện text != null trước khi gọi phương thức trên biến
* Thay System.out.println bằng Logger
* Sử dụng try-with-resources nếu có tài nguyên cần đóng (trong trường hợp mở rộng)

### **Viết lại code sạch hơn**



## **Bài tập 10: OvercatchingExceptionExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Overcatching exception**

* Loại lỗi: **Exception Handling Issue** → bắt exception quá chung

**Lỗi 2: Standard outputs should not be used directly to log anything**

* Loại lỗi: **Code Smell**

**Lỗi 3: Format strings should be used correctly**

* Loại lỗi: **Code Quality Issue**

**Lỗi 4: Không log chi tiết nguyên nhân lỗi**

* Loại lỗi: **Debugging Issue** → gây khó khăn cho việc debug

### **Giải thích tại sao đây là lỗi**

**Overcatching exception:**

* Việc bắt RuntimeException quá chung sẽ che giấu các loại lỗi cụ thể
* Trong ví dụ này, lỗi thực sự là ArrayIndexOutOfBoundsException → nên được bắt đúng loại để xử lý chính xác

**Standard outputs:**

* Sử dụng System.out.println trực tiếp để ghi log không đúng chuẩn, làm cho việc quản lý log không hiệu quả và không phù hợp với các dự án thực tế

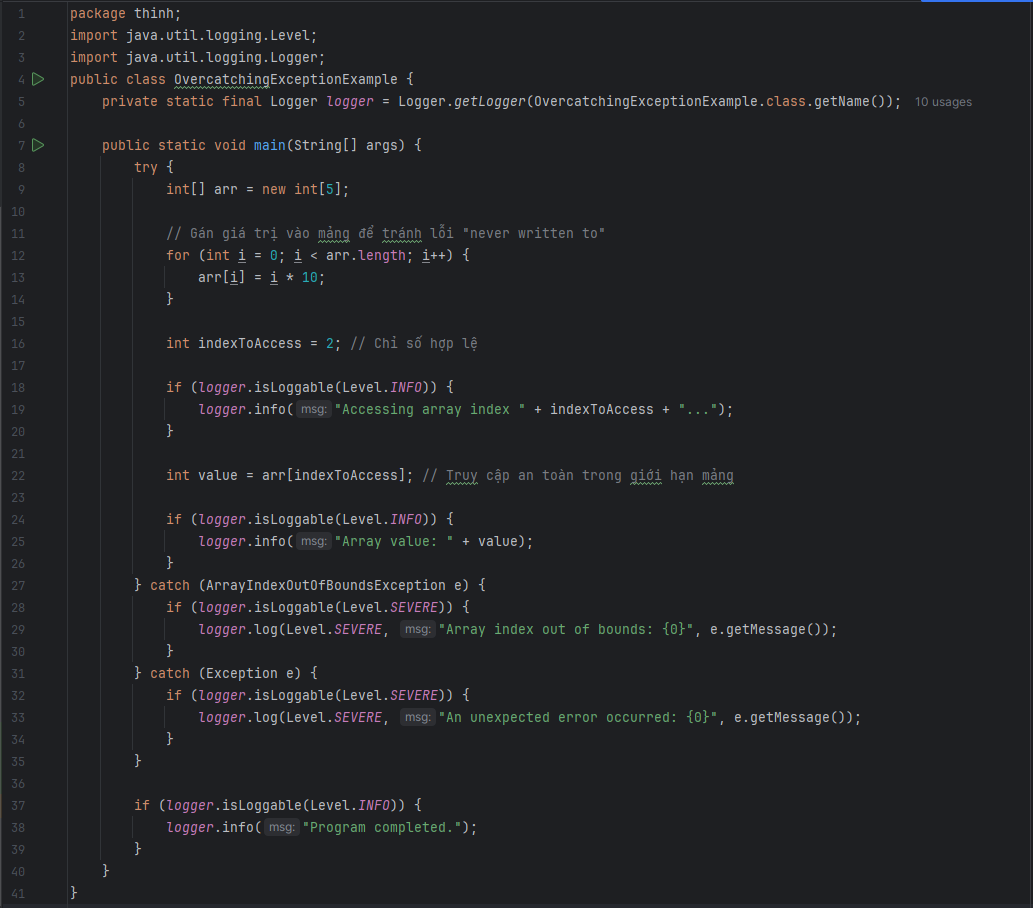
**Không log chi tiết:**

* Việc chỉ ghi "Runtime error" mà không log stack trace hoặc thông tin lỗi cụ thể sẽ gây khó khăn trong việc tìm nguyên nhân lỗi

### **Đề xuất cách sửa**

* Bắt chính xác ngoại lệ ArrayIndexOutOfBoundsException thay vì RuntimeException
* Sử dụng Logger để log lỗi
* Log chi tiết nội dung lỗi và stack trace
* Sử dụng format string đúng chuẩn

### **Viết lại code sạch hơn**



## **Bài tập 11: TightCouplingExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Tightly coupled code**

* Loại lỗi: **Design Pattern Violation** (Code bị ràng buộc chặt chẽ)

**Lỗi 2: Standard outputs should not be used directly to log anything**

* Loại lỗi: **Code Smell**

**Lỗi 3: Vi phạm nguyên tắc Dependency Inversion**

* Loại lỗi: **SOLID Principle Violation** (nguyên lý D trong SOLID)

### **Giải thích tại sao đây là lỗi**

**Tightly coupled code:**

* Lớp Report phụ thuộc trực tiếp vào lớp Printer cụ thể
* Nếu muốn thay đổi cách in (ví dụ: in ra file, in ra logger, in ra web) thì phải sửa code trong Report → rất khó mở rộng và bảo trì

**Standard outputs:**

* Việc sử dụng System.out.println không được chấp nhận trong các hệ thống thực tế vì không kiểm soát được log, không thể phân biệt mức độ log (INFO, DEBUG, ERROR...)

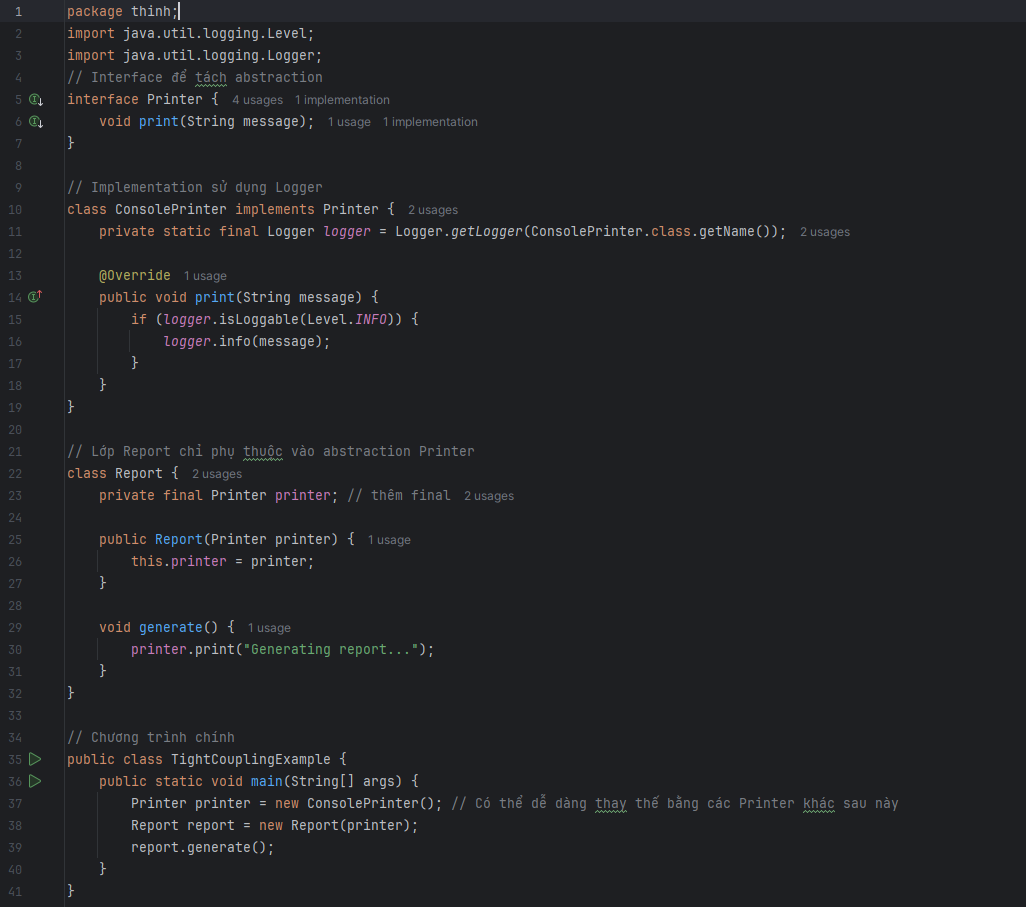
**Vi phạm Dependency Inversion:**

* Report nên phụ thuộc vào abstraction (interface) thay vì phụ thuộc vào cụ thể (Printer class)

### **Đề xuất cách sửa**

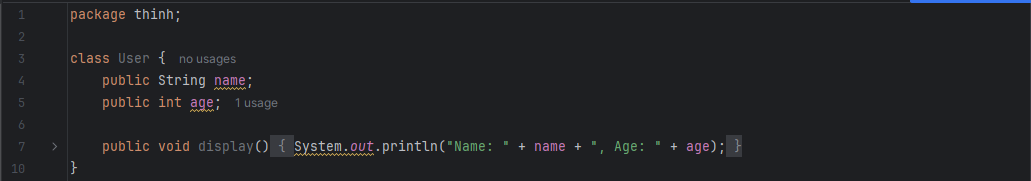
* Tạo một interface Printer để tách biệt abstraction
* Sử dụng **Dependency Injection** để truyền đối tượng Printer từ bên ngoài vào Report
* Thay thế System.out.println bằng Logger chuẩn
* Sử dụng String.format() để định dạng chuỗi chính xác và dễ đọc

### **Viết lại code sạch hơn**



## **Bài tập 12: ViolationOfEncapsulationExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Vi phạm nguyên tắc đóng gói**

* Loại lỗi: **Encapsulation Violation**

**Lỗi 2: Sử dụng trực tiếp System.out.println để log**

* Loại lỗi: **Code Smell** (Standard outputs should not be used directly to log anything)

**Lỗi 3: Format string không được sử dụng đúng cách**

* Loại lỗi: **Code Quality Issue** (Format strings should be used correctly)

### **Giải thích tại sao đây là lỗi**

**Vi phạm đóng gói:**

* Các biến name và age được khai báo là public → Dữ liệu đối tượng có thể bị truy cập và thay đổi trực tiếp từ bên ngoài mà không có sự kiểm soát, dễ gây lỗi và khó bảo trì

**Standard outputs:**

* Việc sử dụng System.out.println không được khuyến khích trong các ứng dụng thực tế vì không kiểm soát được log, không hỗ trợ các cấp độ log, không có khả năng ghi log vào file

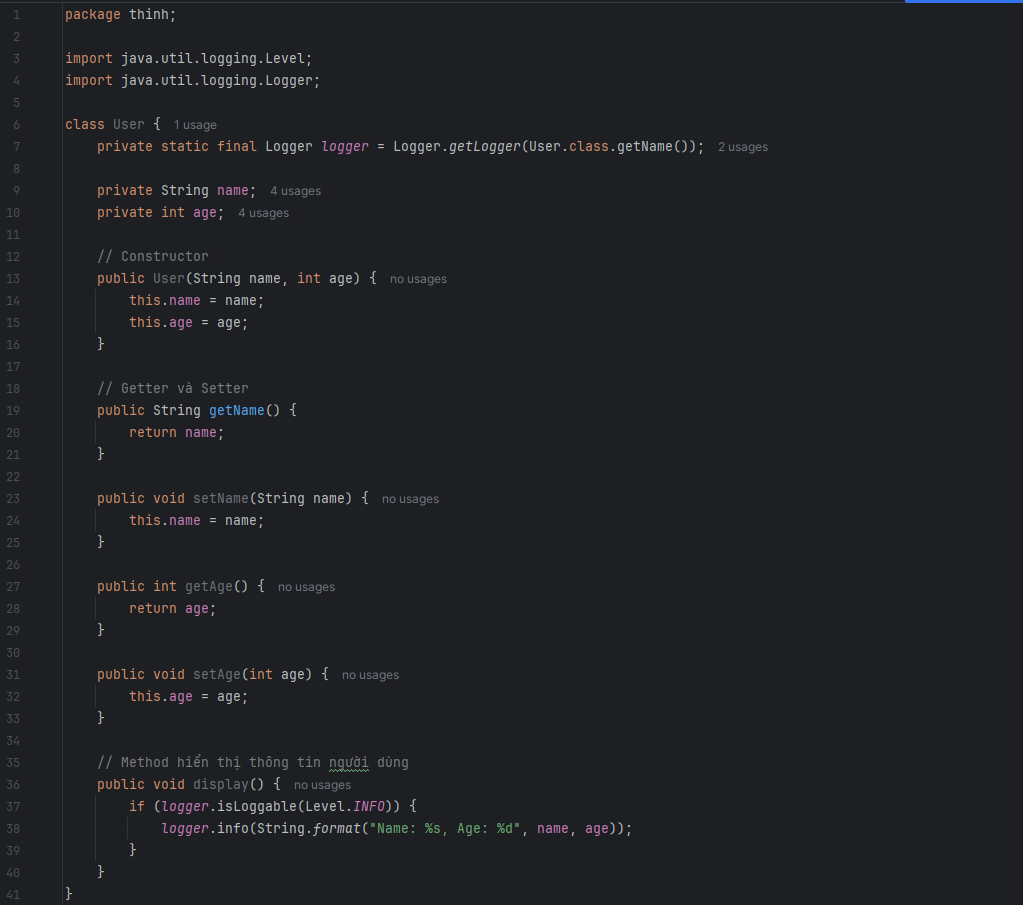
**Format string không chính xác:**

* Việc nối chuỗi thủ công sẽ gây khó đọc, kém hiệu quả so với sử dụng các phương thức format string tiêu chuẩn

### **Đề xuất cách sửa**

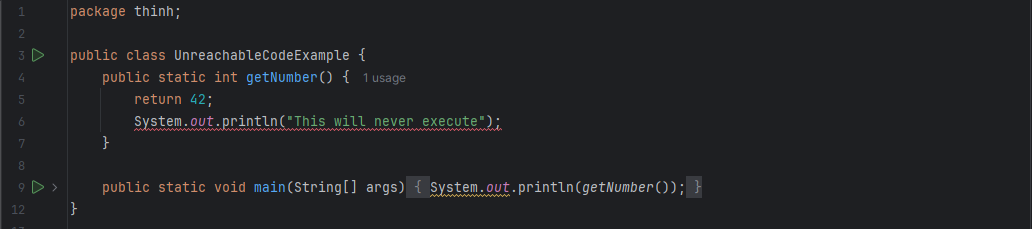
* Đặt các thuộc tính name và age ở chế độ private và cung cấp getter và setter để truy xuất có kiểm soát
* Thay thế System.out.println bằng Logger chuẩn
* Sử dụng String.format() để định dạng chuỗi chính xác và dễ đọc

### **Viết lại code sạch hơn**



## **Bài tập 13: UnreachableCodeExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Unreachable code**

* Loại lỗi: **Compile Error** (Mã không thể thực thi được)

**Lỗi 2: Sử dụng trực tiếp System.out.println để log**

* Loại lỗi: **Code Smell** (Standard outputs should not be used directly to log anything)

**Lỗi 3: Format string không được sử dụng đúng cách**

* Loại lỗi: **Code Quality Issue** (Format strings should be used correctly)

### **Giải thích tại sao đây là lỗi**

**Unreachable code:**

* Trong phương thức getNumber(), lệnh System.out.println("This will never execute"); nằm sau câu lệnh return 42; → Trình biên dịch sẽ báo lỗi vì code sau return sẽ không bao giờ được thực thi

**Standard outputs:**

* Việc sử dụng System.out.println không được khuyến khích vì không hỗ trợ quản lý log chuyên nghiệp, không phân cấp độ log, không ghi được log ra file

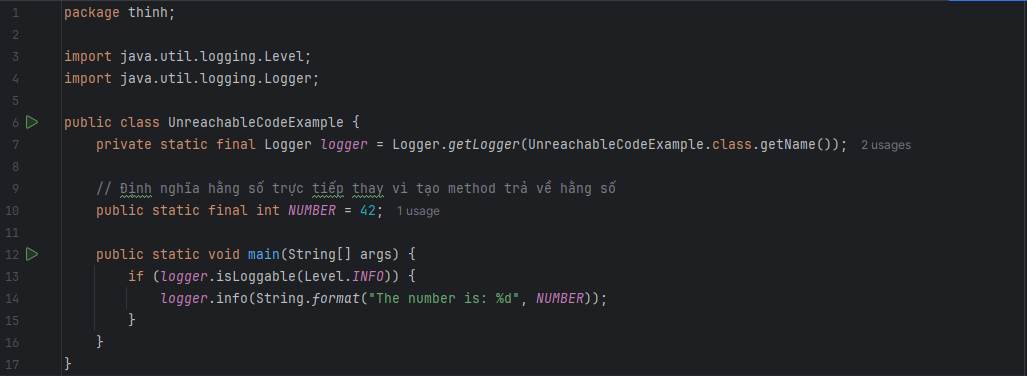
**Format string:**

* Khi log, nên sử dụng phương pháp format string đúng chuẩn thay vì nối chuỗi thủ công

### **Đề xuất cách sửa**

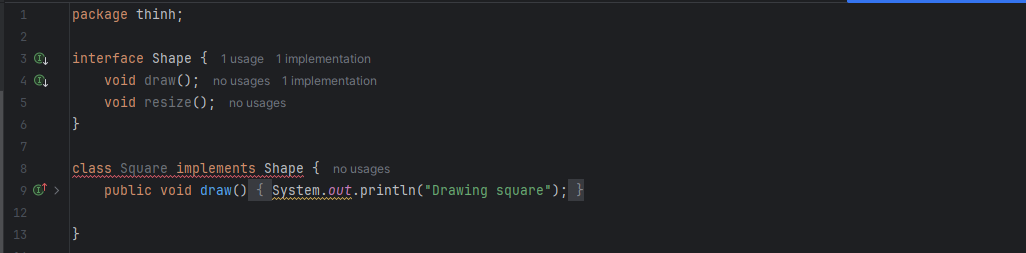
* Xóa đoạn mã unreachable (không thể thực thi)
* Thay thế System.out.println bằng Logger chuẩn
* Sử dụng String.format() khi cần format chuỗi trong log
* Kiểm tra cấp độ log trước khi log nếu có phép tính phức tạp

### **Viết lại code sạch hơn**



## **Bài tập 14: IncompleteInterfaceImplementationExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Không triển khai đầy đủ phương thức của interface**

* Loại lỗi: **Compile-time Error** (Unimplemented Method)

**Lỗi 2: Sử dụng trực tiếp System.out.println để log**

* Loại lỗi: **Code Smell** (Standard outputs should not be used directly to log anything)

**Lỗi 3: Format string không được sử dụng đúng cách**

* Loại lỗi: **Code Quality Issue** (Format strings should be used correctly)

### **Giải thích tại sao đây là lỗi**

**Không triển khai đầy đủ interface:**

* Lớp Square đã implement interface Shape nhưng chỉ viết phương thức draw() và thiếu phương thức resize()
* Điều này sẽ gây lỗi biên dịch vì khi đã implement một interface, tất cả các phương thức trong interface đó bắt buộc phải được triển khai

**Standard outputs:**

* Việc sử dụng System.out.println trong phương thức draw là không tốt trong các hệ thống thực tế vì không quản lý được log và không có khả năng kiểm soát cấp độ log

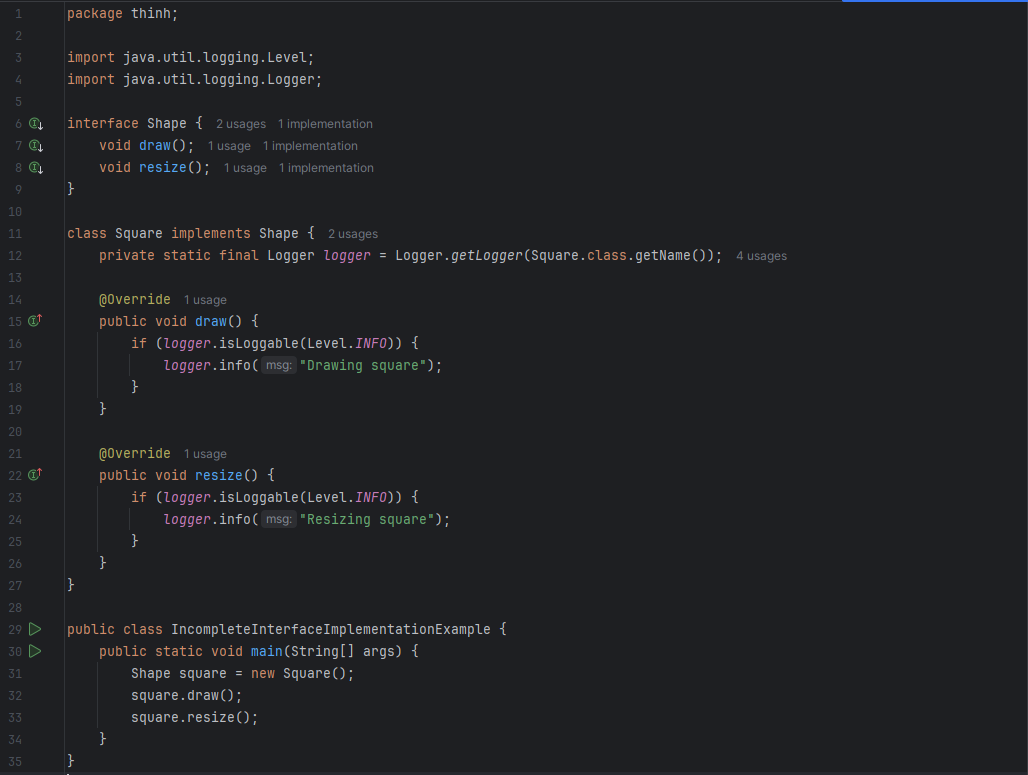
**Format string không được sử dụng:**

* Câu lệnh in ra sử dụng nối chuỗi thủ công, nên thay thế bằng String.format hoặc dùng Logger hỗ trợ format string

### **Đề xuất cách sửa**

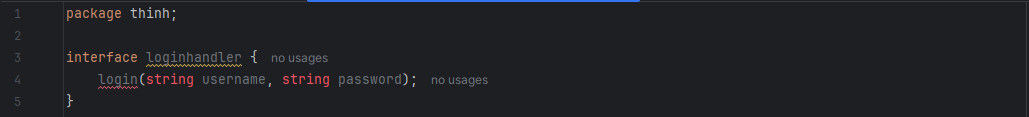
* Triển khai đầy đủ tất cả các phương thức của interface Shape
* Thay thế System.out.println bằng Logger chuẩn
* Sử dụng String.format() hoặc logger hỗ trợ format string

### **Viết lại code sạch hơn**



## **Bài tập 15: InterfaceNamingInconsistencyExample**

### **Đoạn code nguồn ban đầu**



### **Các lỗi phát hiện được (phân loại)**

**Lỗi 1: Không khai báo kiểu trả về cho phương thức trong interface**

* Loại lỗi: **Compile-time Error**

**Lỗi 2: Sai quy tắc đặt tên trong Java**

* Loại lỗi: **Naming Convention Violation**

**Lỗi 3: Tên phương thức và tên interface nên tuân theo quy ước đặt tên chuẩn**

* Loại lỗi: **Code Quality Issue** để đảm bảo code clean

**Lỗi 4: Chưa sử dụng logger, sử dụng không đúng format string**

* Loại lỗi: **Code Smell** (nếu mở rộng trong phương thức thực thi sau này)

### **Giải thích tại sao đây là lỗi**

**Không có kiểu trả về:**

* Trong Java, khi khai báo phương thức trong interface, phải có kiểu trả về (ví dụ: void, boolean, int,...) nếu không sẽ gây lỗi biên dịch

**Sai quy tắc đặt tên:**

* **Tên interface:** loginhandler → Viết thường là sai. Theo convention, interface nên viết PascalCase → phải là LoginHandler
* **Tên phương thức:** login là đúng nhưng các tham số string phải viết hoa chữ S → String

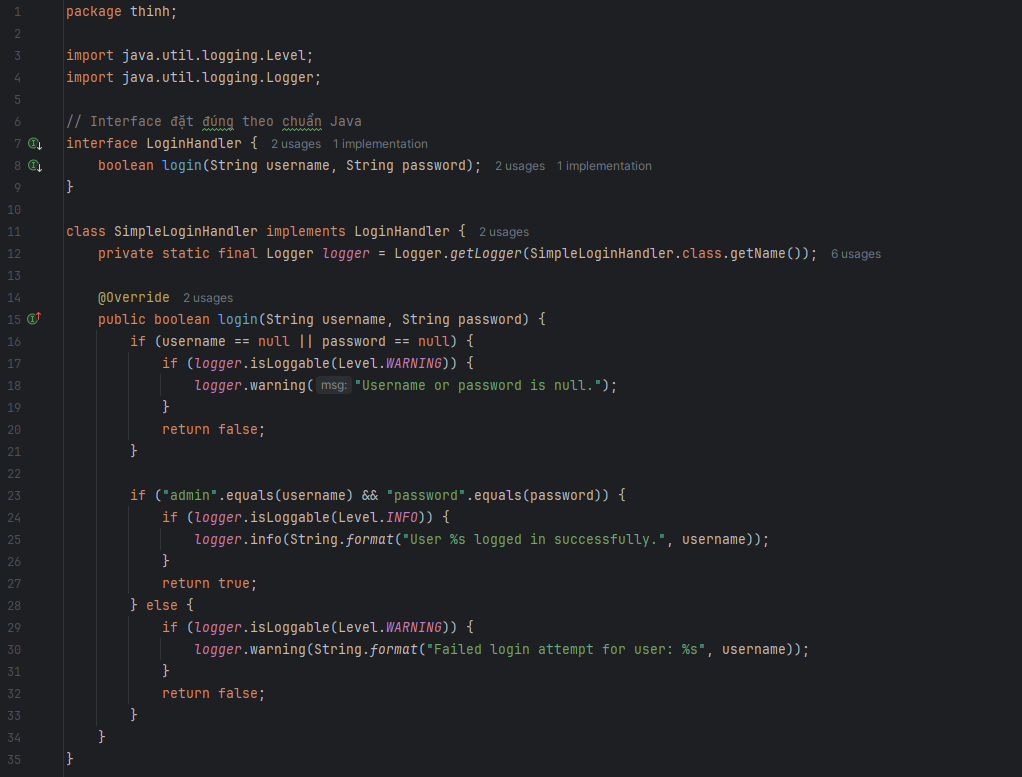
**Logger:**

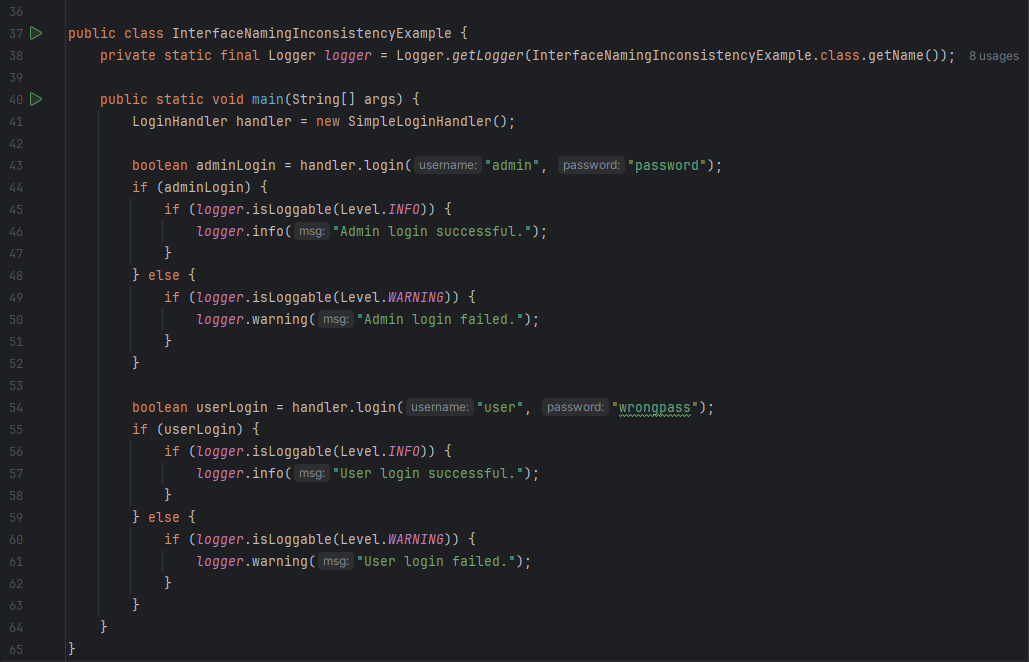
* Trong các dự án thực tế, nếu interface có log, không được sử dụng System.out.println, mà phải dùng logger

### **Đề xuất cách sửa**

* Sửa tên interface theo PascalCase → LoginHandler
* Thêm kiểu trả về cho phương thức → ví dụ: boolean login(String username, String password);
* Sửa lại kiểu dữ liệu tham số: string → String
* Khi implement interface, sử dụng Logger để log thông tin

### **Viết lại code sạch hơn**





## **Kết luận**

Qua việc phân tích 15 bài tập về các lỗi thường gặp trong lập trình, chúng ta có thể thấy rằng:

### **Các nhóm lỗi chính:**

1. **Security vulnerabilities:** Hardcoded credentials, SQL injection, Path traversal
2. **Code smells:** Sử dụng System.out.println, không sử dụng logging framework
3. **Runtime errors:** Resource leak, Null pointer, Array index out of bounds
4. **Compile errors:** Unimplemented interface, Unreachable code
5. **Design pattern violations:** Tight coupling, Encapsulation violation
6. **Exception handling issues:** Catching generic exceptions, Overcatching

### **Nguyên tắc cần tuân thủ:**

* **Bảo mật:** Không hardcode credentials, sử dụng PreparedStatement
* **Logging:** Sử dụng logging framework thay vì System.out.println
* **Exception handling:** Bắt exception cụ thể, không bắt quá chung
* **OOP principles:** Tuân thủ encapsulation, sử dụng interface và dependency injection
* **Code quality:** Kiểm tra null, đóng resources, sử dụng format string