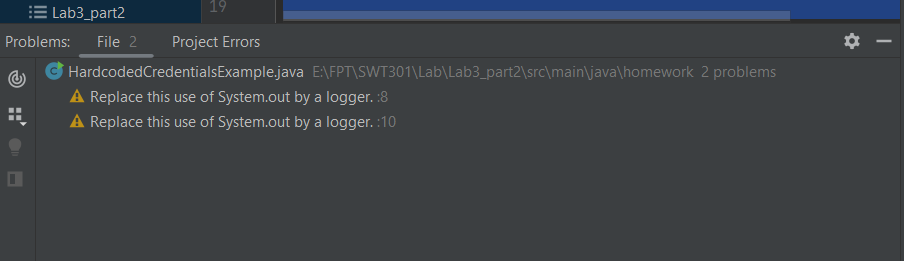
# **Homeworks:**

1. **HardcodedCredentialsExample**
   * **Đoạn code nguồn ban đầu**

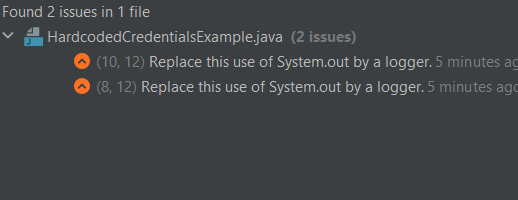
package homework;  
  
public class HardcodedCredentialsExample {  
 public static void main(String[] args) {  
 String username = "admin";  
 String password = "123456"; // hardcoded password  
 if(*authenticate*(username, password)) {  
 System.*out*.println("Access granted");  
 } else {  
 System.*out*.println("Access denied");  
 }  
 }  
  
 private static boolean authenticate(String user, String pass) {  
 // Dummy authentication logic  
 return user.equals("admin") && pass.equals("123456");  
 }  
}

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

****

* + **Giải thích tại sao đây là lỗi**

Lý do:

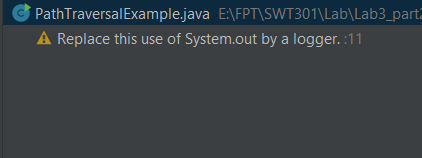
* Dùng System.out.println là cách đơn giản để in ra màn hình, nhưng:
* Không chuyên nghiệp cho ứng dụng thật.
* Không kiểm soát được cấp độ log (info, error, warning).
* Không ghi log ra file hay hệ thống log trung tâm.
  + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

package homework;  
  
import java.util.logging.Logger;  
  
public class HardcodedCredentialsExample {  
 private static final Logger *logger* = Logger.*getLogger*(HardcodedCredentialsExample.class.getName());  
  
 public static void main(String[] args) {  
 String username = "admin";  
 String password = "123456"; // hardcoded password  
 if (*authenticate*(username, password)) {  
 *logger*.info("Access granted");  
 } else {  
 *logger*.warning("Access denied");  
 }  
 }  
  
 private static boolean authenticate(String user, String pass) {  
 // Dummy authentication logic  
 return user.equals("admin") && pass.equals("123456");  
 }  
}

1. **PathTraversalExample**
   * Đoạn code nguồn ban đầu

package homework;  
  
import java.io.\*;  
  
public class PathTraversalExample {  
 public static void main(String[] args) throws IOException {  
 String userInput = "../secret.txt";  
 File file = new File(userInput);  
 if (file.exists()) {  
 BufferedReader reader = new BufferedReader(new FileReader(file));  
 System.*out*.println("Reading file: " + file.getPath());  
 reader.close();  
 }  
 }  
}

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

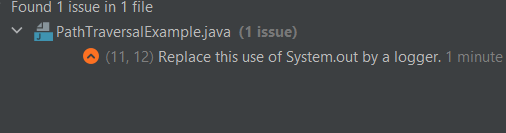


* **Giải thích tại sao đây là lỗi**

System.out.println bị cảnh báo là không phù hợp cho ứng dụng thực tế.

Dùng Logger thay thế giúp dễ quản lý log, ghi ra file, phân loại log, và bảo mật hơn.

* + **Đề xuất cách sửa**



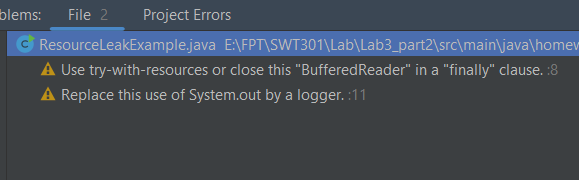
* + **Viết lại code sạch hơn**

package homework;  
  
import java.io.\*;  
import java.util.logging.Logger;  
  
public class PathTraversalExample {  
 private static final Logger *logger* = Logger.*getLogger*(PathTraversalExample.class.getName());  
  
 public static void main(String[] args) throws IOException {  
 String userInput = "../secret.txt";  
 File file = new File(userInput);  
 if (file.exists()) {  
 BufferedReader reader = new BufferedReader(new FileReader(file));  
 *logger*.info("Reading file: " + file.getPath());  
 reader.close();  
 }  
 }  
}

1. **ResourceLeakExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
import java.io.\*;  
  
public class ResourceLeakExample {  
 public static void main(String[] args) {  
 try {  
 BufferedReader reader = new BufferedReader(new FileReader("data.txt"));  
 String line;  
 while ((line = reader.readLine()) != null) {  
 System.*out*.println(line);  
 }  
   
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

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

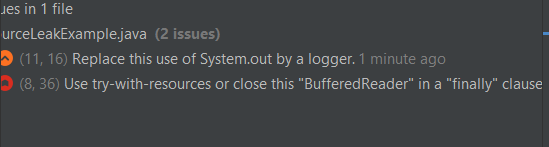


* + **Giải thích tại sao đây là lỗi**

**Lý do:**  
Tạo BufferedReader nhưng **không đóng nó** (reader.close()). Nếu không đóng, có thể gây **rò rỉ tài nguyên** (resource leak), đặc biệt khi chương trình đọc nhiều file.

System.out.println là cách in đơn giản, không phù hợp cho ứng dụng thực tế vì:

* Không phân loại mức độ thông báo (info, warning, error).
* Không ghi log ra file hoặc hệ thống log.
* Khó kiểm soát khi ứng dụng lớn.
  + **Đề xuất cách sửa**

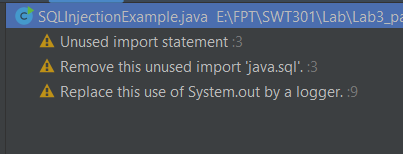


* + **Viết lại code sạch hơn**

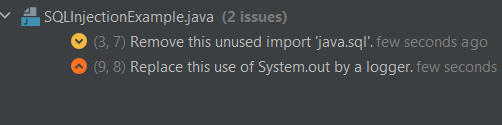
package homework;  
  
import java.io.\*;  
import java.util.logging.Logger;  
  
public class ResourceLeakExample {  
 private static final Logger *logger* = Logger.*getLogger*(ResourceLeakExample.class.getName());  
  
 public static void main(String[] args) {  
 try (BufferedReader reader = new BufferedReader(new FileReader("data.txt"))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 *logger*.info(line);  
 }  
 } catch (IOException e) {  
 *logger*.severe("Error reading file: " + e.getMessage());  
 }  
 }  
}

1. **SQLInjectionExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
import java.sql.\*;  
  
public class SQLInjectionExample {  
 public static void main(String[] args) {  
 String userInput = "' OR '1'='1";  
 String query = "SELECT \* FROM users WHERE username = '" + userInput + "'";  
 System.*out*.println("Executing query: " + query);  
 }  
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

|  |  |
| --- | --- |
| **Cảnh báo** | **Nguyên nhân** |
| **Unused import statement** | Bạn import java.sql.\*; nhưng không dùng bất kỳ class nào từ đó (ví dụ: Connection, Statement, v.v.). |
| **Remove this unused import 'java.sql'** | Khuyến nghị xóa dòng import java.sql.\*; để mã sạch hơn. |
| **Replace System.out by a logger** | Không nên dùng System.out.println, thay bằng Logger để dễ quản lý log và đúng chuẩn Java. |

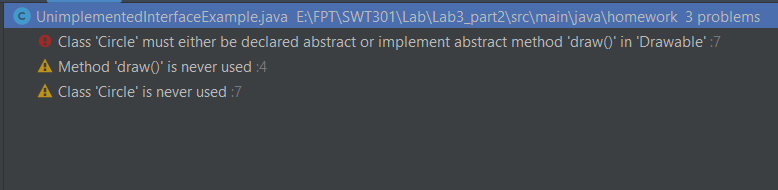
* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

package homework;  
  
import java.util.logging.Logger;  
  
public class SQLInjectionExample {  
 private static final Logger *logger* = Logger.*getLogger*(SQLInjectionExample.class.getName());  
  
 public static void main(String[] args) {  
 String userInput = "' OR '1'='1";  
 String query = "SELECT \* FROM users WHERE username = '" + userInput + "'";  
 *logger*.warning("Executing query: " + query); // warning vì chứa dữ liệu đầu vào  
 }  
}

1. **UnimplementedInterfaceExample**

* **Đoạn code nguồn ban đầu**

package homework;  
  
interface Drawable {  
 void draw();  
}  
  
class Circle implements Drawable {  
 // Missing draw() implementation → compile error  
}

* **Các lỗi phát hiện được (phân loại)**
* **Giải thích tại sao đây là lỗi và Đề xuất cách sửa**

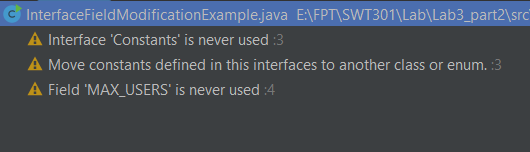
|  |  |  |
| --- | --- | --- |
| **Vấn đề gốc** | **Vì sao sai** | **Cách đã sửa** |
| Không implement draw() | Vi phạm hợp đồng của interface | Thêm @Override public void draw() |
| Dùng System.out.println | Không chuyên nghiệp, không theo chuẩn logging | Dùng Logger để ghi thông điệp |
| Không sử dụng class, method | Dễ bị cảnh báo “unused” | Gọi shape.draw() trong main() |

* **Viết lại code sạch hơn**

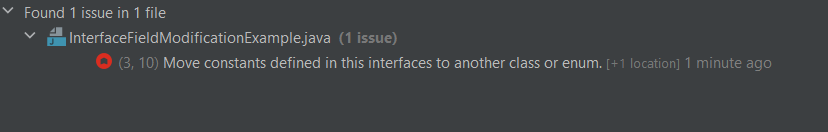
package homework;  
  
import java.util.logging.Logger;  
  
// Interface với phương thức draw  
interface Drawable {  
 void draw();  
}  
  
// Circle triển khai Drawable  
class Circle implements Drawable {  
 private static final Logger *logger* = Logger.*getLogger*(Circle.class.getName());  
  
 @Override  
 public void draw() {  
 *logger*.info("Drawing a circle");  
 }  
}  
  
// Lớp chính để chạy chương trình  
public class UnimplementedInterfaceExample {  
 public static void main(String[] args) {  
 Drawable shape = new Circle(); // tạo đối tượng Circle thông qua interface  
 shape.draw(); // gọi phương thức draw, ghi log  
 }  
}

1. **InterfaceFieldModificationExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
interface Constants {  
 int *MAX\_USERS* = 100;  
}  
  
public class InterfaceFieldModificationExample {  
 public static void main(String[] args) {  
 // Constants.MAX\_USERS = 200; // Compile-time error  
 }  
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

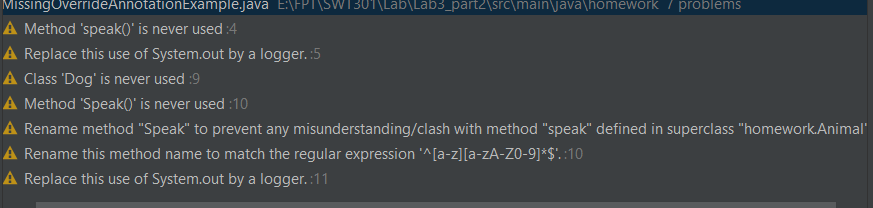
|  |  |
| --- | --- |
| **Cảnh báo** | **Giải thích** |
| **Interface 'Constants' is never used** | Interface khai báo nhưng không được gọi ở đâu. |
| **Move constants defined in this interface to another class or enum** | Java **không khuyến khích dùng interface để chứa hằng số**. Nên dùng final class hoặc enum. |
| **Field 'MAX\_USERS' is never used** | Hằng số được khai báo nhưng chưa bao giờ truy cập → bị cảnh báo là dư thừa. |

* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

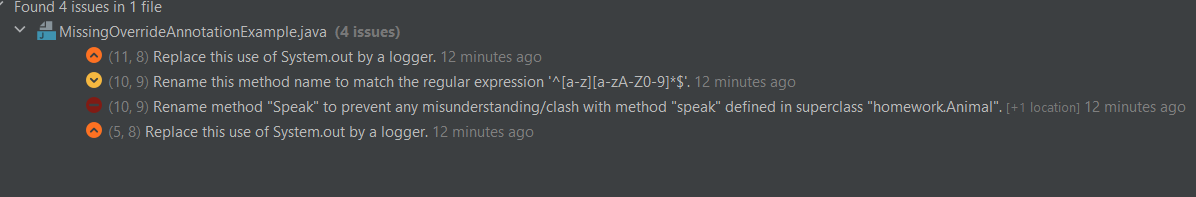
package homework;  
  
import java.util.logging.Logger;  
  
// Class chứa hằng số  
final class AppConstants {  
 public static final int *MAX\_USERS* = 100;  
  
 private AppConstants() {  
 // ngăn tạo đối tượng  
 }  
}  
  
// Class chính sử dụng Logger  
public class InterfaceFieldModificationExample {  
 private static final Logger *logger* = Logger.*getLogger*(InterfaceFieldModificationExample.class.getName());  
  
 public static void main(String[] args) {  
 int max = AppConstants.*MAX\_USERS*;  
 *logger*.info("Max allowed users: " + max);  
 }  
}

1. **MissingOverrideAnnotationExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
class Animal {  
 void speak() {  
 System.*out*.println("Animal speaks");  
 }  
}  
  
class Dog extends Animal {  
 void Speak() {   
 System.*out*.println("Dog barks");  
 }  
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

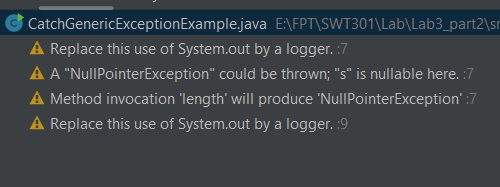
|  |  |
| --- | --- |
| **Cảnh báo** | **Giải thích** |
| Method speak() is never used | Hàm không được gọi ở đâu cả. |
| Replace this use of System.out by a logger | Clean code khuyên dùng Logger. |
| Class 'Dog' is never used | Bạn khai báo lớp Dog nhưng không tạo đối tượng hoặc gọi trong chương trình. |
| Method 'Speak()' is never used | Phương thức viết hoa Speak() không theo quy tắc đặt tên. |
| Rename method "Speak" | Tên Speak gây nhầm lẫn với speak() trong lớp cha Animal. |
| Rename this method name to match '^ [a-z][a-zA-Z0-9]\*$' | Tên phương thức phải bắt đầu bằng chữ thường (camelCase). |

* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

package homework;  
  
import java.util.logging.Logger;  
  
// Lớp cha  
class Animal {  
 void speak() {  
 Logger.*getLogger*(Animal.class.getName()).info("Animal speaks");  
 }  
}  
  
// Lớp con  
class Dog extends Animal {  
 private static final Logger *logger* = Logger.*getLogger*(Dog.class.getName());  
  
 @Override  
 void speak() {  
 *logger*.info("Dog barks");  
 }  
}  
  
// Class chính để kiểm tra  
public class MissingOverrideAnnotationExample {  
 public static void main(String[] args) {  
 Animal animal = new Dog();  
 animal.speak(); // In ra: Dog barks (nếu override đúng)  
 }  
}

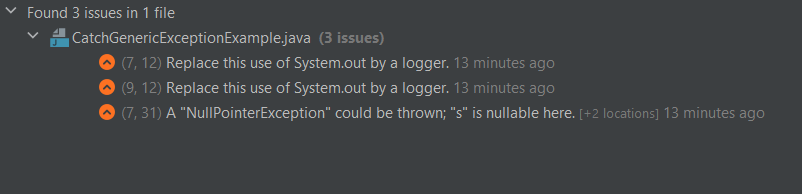
1. **CatchGenericExceptionExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
public class CatchGenericExceptionExample {  
 public static void main(String[] args) {  
 try {  
 String s = null;  
 System.*out*.println(s.length());  
 } catch (Exception e) {  
 System.*out*.println("Something went wrong");   
 }  
 }  
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

|  |  |
| --- | --- |
| **Cảnh báo** | **Nguyên nhân** |
| **Replace System.out by a logger** | Bạn dùng System.out.println → nên thay bằng Logger |
| **"s" is nullable → NullPointerException** | Biến s = null, nhưng lại gọi s.length() → chắc chắn lỗi |
| **length() sẽ gây NullPointerException** | Do không kiểm tra s != null trước khi gọi .length() |

* + **Đề xuất cách sửa**

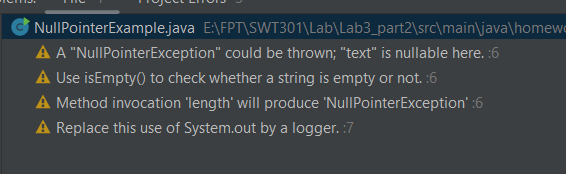
****

* + **Viết lại code sạch hơn**

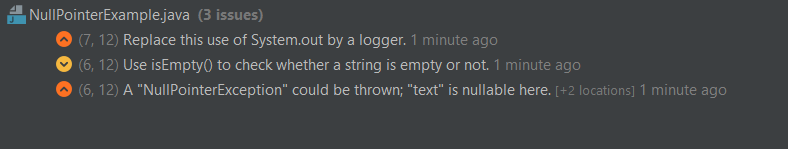
package homework;  
  
import java.util.logging.Logger;  
  
public class CatchGenericExceptionExample {  
 private static final Logger *logger* = Logger.*getLogger*(CatchGenericExceptionExample.class.getName());  
  
 public static void main(String[] args) {  
 String s = *getNullableString*(); // nhận dữ liệu có thể null  
  
 try {  
 if (s != null) {  
 *logger*.info("Length: " + s.length());  
 } else {  
 *logger*.warning("String is null, cannot call length()");  
 }  
 } catch (NullPointerException e) {  
 *logger*.severe("Unexpected NullPointerException: " + e.getMessage());  
 }  
 }  
  
 private static String getNullableString() {  
 // Mô phỏng dữ liệu có thể null, ví dụ từ DB hoặc input  
 return Math.*random*() > 0.5 ? "hello" : null;  
 }  
}

1. **NullPointerExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
public class NullPointerExample {  
 public static void main(String[] args) {  
 String text = null;  
 if (text.length() > 0) {   
 System.*out*.println("Text is not empty");  
 }  
 }  
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

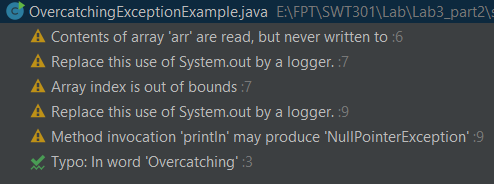
|  |  |
| --- | --- |
| **Cảnh báo** | **Giải thích** |
| "text" is nullable here | Bạn gán text = null rồi gọi text.length() → sẽ gây **NullPointerException** |
| Use isEmpty() to check whether a string is empty or not | Nên dùng text.isEmpty() (sau khi kiểm tra text != null) để kiểm tra chuỗi rỗng |
| System.out | Không nên dùng System.out.println → nên dùng Logger theo chuẩn chuyên nghiệp |

* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

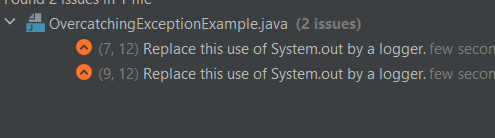
package homework;  
  
import java.util.logging.Logger;  
  
public class NullPointerExample {  
 private static final Logger *logger* = Logger.*getLogger*(NullPointerExample.class.getName());  
  
 public static void main(String[] args) {  
 String text = *getNullableText*(); // mô phỏng đầu vào có thể null  
  
 if (text != null && !text.isEmpty()) {  
 *logger*.info("Text is not empty");  
 } else if (text == null) {  
 *logger*.warning("Text is null");  
 } else {  
 *logger*.info("Text is empty");  
 }  
 }  
  
 private static String getNullableText() {  
 // Mô phỏng nguồn dữ liệu: đôi khi null, đôi khi có chuỗi  
 return Math.*random*() > 0.5 ? "hello" : null;  
 }  
}

1. **OvercatchingExceptionExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
public class OvercatchingExceptionExample {  
 public static void main(String[] args) {  
 try {  
 int[] arr = new int[5];  
 System.*out*.println(arr[10]);   
 } catch (RuntimeException e) {  
 System.*out*.println("Runtime error");   
 }  
 }  
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

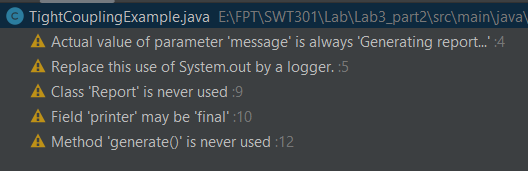
|  |  |
| --- | --- |
| **Cảnh báo** | **Nguyên nhân** |
| Contents of array 'arr' are read, but never written to | Mảng arr được đọc tại arr[10], nhưng không có giá trị nào được gán. |
| Array index is out of bounds | Truy cập phần tử arr[10] trong mảng arr = new int[5] → lỗi ArrayIndexOutOfBoundsException vì chỉ có chỉ số từ 0 đến 4. |
| Overcatching: catch RuntimeException | Bạn bắt RuntimeException chung chung → nên bắt cụ thể ArrayIndexOutOfBoundsException. |
| System.out.println(...) | Cần thay bằng Logger theo clean code. |
| Method println may produce NullPointerException | Nếu System.out là null (hiếm), có thể gây lỗi (phân tích tĩnh báo vậy) |

* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

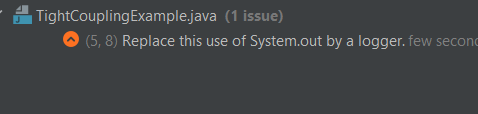
package homework;  
  
import java.util.logging.Logger;  
  
public class OvercatchingExceptionExample {  
 private static final Logger *logger* = Logger.*getLogger*(OvercatchingExceptionExample.class.getName());  
  
 public static void main(String[] args) {  
 try {  
 int[] arr = new int[5];  
 // Gán giá trị để tránh cảnh báo "never written to"  
 for (int i = 0; i < arr.length; i++) {  
 arr[i] = i \* 10;  
 }  
  
 *logger*.info("Accessing element at index 10...");  
 *logger*.info("Value: " + arr[2]); // ví dụ với chỉ số hợp lệ  
 } catch (ArrayIndexOutOfBoundsException e) {  
 *logger*.severe("Array index is out of bounds: " + e.getMessage());  
 }  
 }  
}

1. **TightCouplingExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
class Printer {  
 void print(String message) {  
 System.*out*.println(message);  
 }  
}  
  
class Report {  
 private Printer printer = new Printer(); // tightly coupled  
  
 void generate() {  
 printer.print("Generating report...");  
 }  
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

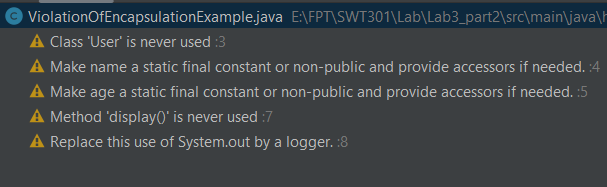
|  |  |
| --- | --- |
| **Cảnh báo** | **Giải thích** |
| **Actual value of parameter 'message' is always 'Generating report...'** | Bạn luôn truyền cùng một chuỗi "Generating report...", nên tham số message đang dư thừa. |
| **Replace System.out by a logger** | Clean code khuyến nghị dùng Logger thay cho System.out.println. |
| **Class 'Report' is never used** | Class được khai báo nhưng không có đoạn code nào sử dụng hoặc khởi tạo nó. |
| **Field 'printer' may be final** | Biến printer không thay đổi sau khi gán → nên thêm từ khóa final để rõ ý định bất biến. |
| **Method 'generate()' is never used** | Phương thức được khai báo nhưng chưa được gọi ở đâu trong chương trình. |

* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

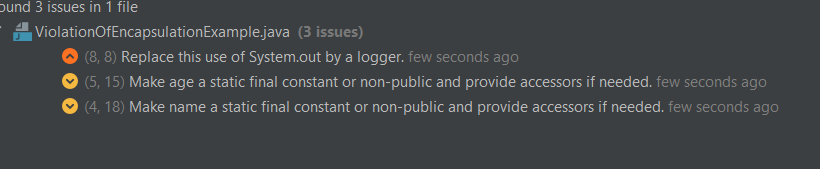
package homework;  
  
import java.util.logging.Logger;  
  
// Interface để giảm coupling  
interface OutputDevice {  
 void print(String message);  
}  
  
// Triển khai cụ thể cho OutputDevice  
class Printer implements OutputDevice {  
 private static final Logger *logger* = Logger.*getLogger*(Printer.class.getName());  
  
 @Override  
 public void print(String message) {  
 *logger*.info(message);  
 }  
}  
  
// Lớp Report giờ phụ thuộc vào abstraction OutputDevice  
class Report {  
 private final OutputDevice printer;  
  
 public Report(OutputDevice printer) {  
 this.printer = printer;  
 }  
  
 public void generate() {  
 printer.print("Generating report...");  
 }  
}  
  
// Lớp main để chạy  
public class TightCouplingExample {  
 public static void main(String[] args) {  
 OutputDevice printer = new Printer();  
 Report report = new Report(printer);  
 report.generate(); // gọi để in report  
 }  
}

1. **ViolationOfEncapsulationExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
class User {  
 public String name;   
 public int age;  
  
 public void display() {  
 System.*out*.println("Name: " + name + ", Age: " + age);  
 }  
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

|  |  |
| --- | --- |
| **Cảnh báo** | **Giải thích** |
| **Class 'User' is never used** | Khai báo class User nhưng không tạo đối tượng hay dùng nó trong chương trình. |
| **Make name/age static final or non-public and provide accessors** | Trường name, age đang là public → vi phạm nguyên tắc đóng gói (encapsulation). Nên để private và cung cấp getter nếu cần. |
| **Method 'display()' is never used** | Hàm display() không được gọi ở đâu cả. |
| **Replace this use of System.out by a logger** | Nên dùng Logger thay vì System.out.println. |

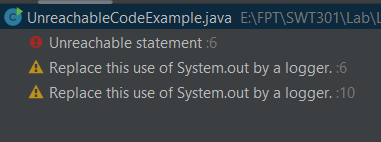
* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

package homework;  
  
import java.util.logging.Logger;  
  
// Class User tuân thủ nguyên tắc đóng gói  
class User {  
 private static final Logger *logger* = Logger.*getLogger*(User.class.getName());  
  
 private final String name;  
 private final int age;  
  
 // Constructor  
 public User(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 // Getter  
 public String getName() {  
 return name;  
 }  
  
 // Hiển thị thông tin  
 public void display() {  
 *logger*.info("Name: " + name + ", Age: " + age);  
 }  
}  
  
// Class chính để sử dụng User  
public class ViolationOfEncapsulationExample {  
 public static void main(String[] args) {  
 User user = new User("Alice", 25);  
 user.display();  
 }  
}

1. **UnreachableCodeExample**
   * **Đoạn code nguồn ban đầu**

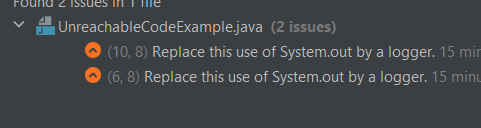
package homework;  
  
public class UnreachableCodeExample {  
 public static int getNumber() {  
 return 42;  
 System.*out*.println("This will never execute");  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println(*getNumber*());  
 }  
}

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



* + **Giải thích tại sao đây là lỗi**

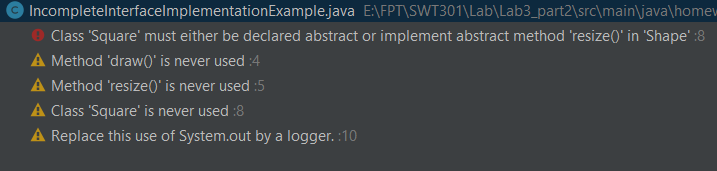
|  |  |
| --- | --- |
| **Cảnh báo / Lỗi** | **Giải thích** |
| **Unreachable statement (dòng 6)** | Bạn có dòng System.out.println(...) nằm sau return nên Java báo lỗi vì dòng đó không bao giờ được chạy. |
| **Replace System.out by a logger** | Clean code không dùng System.out.println mà dùng Logger thay thế. |

* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

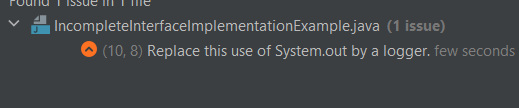
package homework;  
  
import java.util.logging.Logger;  
  
public class UnreachableCodeExample {  
 private static final Logger *logger* = Logger.*getLogger*(UnreachableCodeExample.class.getName());  
  
 public static int getNumber() {  
 // Nếu cần log trước khi return  
 *logger*.info("Preparing to return number");  
 return 42;  
 }  
  
 public static void main(String[] args) {  
 *logger*.info("Returned number: " + *getNumber*());  
 }  
}

1. **IncompleteInterfaceImplementationExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
interface Shape {  
 void draw();  
 void resize();  
}  
  
class Square implements Shape {  
 public void draw() {  
 System.*out*.println("Drawing square");  
 }  
   
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

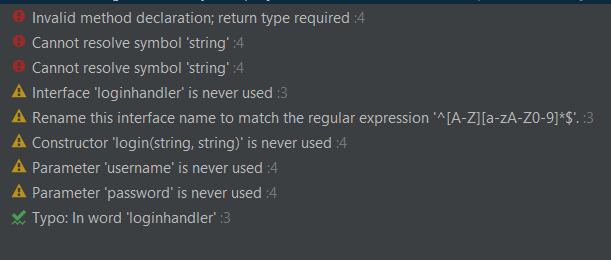
|  |  |
| --- | --- |
| **Cảnh báo / Lỗi** | **Nguyên nhân** |
| **Class 'Square' must either be declared abstract or implement abstract method 'resize()' in 'Shape'** | Lớp Square đã implements Shape nhưng **chưa viết lại (override) phương thức resize()**. |
| **Method 'draw()' is never used** | Hàm được định nghĩa nhưng **không được gọi trong mã**. |
| **Class 'Square' is never used** | Lớp Square không được gọi hoặc khởi tạo ở đâu. |
| **Replace this use of System.out by a logger** | Không nên dùng System.out.println, thay bằng Logger để đúng chuẩn clean code. |

* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

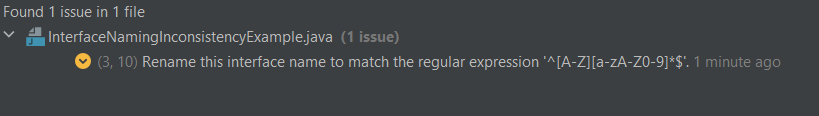
package homework;  
  
import java.util.logging.Logger;  
  
// Interface Shape với 2 phương thức  
interface Shape {  
 void draw();  
 void resize();  
}  
  
// Lớp Square triển khai đầy đủ interface  
class Square implements Shape {  
 private static final Logger *logger* = Logger.*getLogger*(Square.class.getName());  
  
 @Override  
 public void draw() {  
 *logger*.info("Drawing square");  
 }  
  
 @Override  
 public void resize() {  
 *logger*.info("Resizing square");  
 }  
}  
  
// Class chính để sử dụng  
public class IncompleteInterfaceImplementationExample {  
 public static void main(String[] args) {  
 Shape shape = new Square();  
 shape.draw();  
 shape.resize();  
 }  
}

1. **InterfaceNamingInconsistencyExample**
   * **Đoạn code nguồn ban đầu**

package homework;  
  
interface loginhandler {  
 login(string username, string password);  
}

* + **Các lỗi phát hiện được (phân loại)**
  + **Giải thích tại sao đây là lỗi**

|  |  |
| --- | --- |
| **Lỗi / Cảnh báo** | **Nguyên nhân** |
| Invalid method declaration; return type required | Trong interface, bạn đã khai báo phương thức login(...) mà **không có kiểu trả về** (ví dụ: void, boolean, v.v.) |
| Cannot resolve symbol 'string' | Java không có từ khóa string (phải viết đúng là String, có chữ S hoa) |
| Interface 'loginhandler' is never used | Giao diện này không được gọi ở đâu trong mã. |
| Rename interface to match regex '^ [A-Z][a-zA-Z0-9]\*$' | Tên interface phải bắt đầu bằng chữ in hoa theo quy tắc Java → loginhandler sai quy tắc |
| Constructor 'login(...)' is never used | Cảnh báo không quan trọng nếu interface chưa được sử dụng, sẽ hết khi bạn tạo class implements |

* + **Đề xuất cách sửa**
  + **Viết lại code sạch hơn**

package homework;  
  
import java.util.logging.Logger;  
  
// Interface chuẩn tên hóa  
interface LoginHandler {  
 boolean login(String username, String password);  
}  
  
// Class implement interface  
class SimpleLogin implements LoginHandler {  
  
  
 @Override  
 public boolean login(String username, String password) {  
 return "admin".equals(username) && "123".equals(password);  
 }  
}  
  
// Class chính sử dụng LoginHandler  
public class InterfaceNamingInconsistencyExample {  
 private static final Logger *logger* = Logger.*getLogger*(InterfaceNamingInconsistencyExample.class.getName());  
  
 public static void main(String[] args) {  
 LoginHandler handler = new SimpleLogin();  
  
 boolean success = handler.login("admin", "123");  
  
 if (success) {  
 *logger*.info("Login successful");  
 } else {  
 *logger*.info("Login failed");  
 }  
 }  
}