Bai 1:

Code ban dau:

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

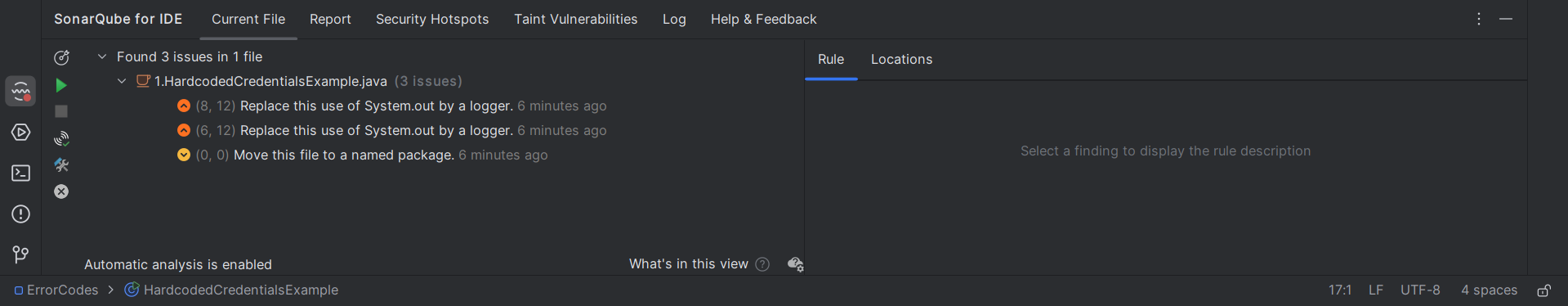
return user.equals("admin") && pass.equals("123456");

}

}

Lỗi bị Sonar phát hiện

| **STT** | **Nội dung lỗi** | **Giải thích** | **Phân loại** |
| --- | --- | --- | --- |
| 1 | Replace this use of System.out by a logger. (dòng 8) | Không nên dùng System.out.println() trong code thực tế. Cần dùng logging framework (như java.util.logging, SLF4J, Log4j...) để dễ dàng quản lý log, phân cấp log, redirect log... | Coding Style / Best Practice |
| 2 | Replace this use of System.out by a logger. (dòng 6) | Cùng lỗi như trên. | Coding Style / Best Practice |
| 3 | Move this file to a named package. (dòng 0) | Class đang không thuộc package nào. Java chuẩn khuyến khích tổ chức code theo package (ví dụ: com.myapp.security). | Coding Style / Structure |



Code hoàn chỉnh:

package com.example.security;  
  
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";  
  
 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");  
 }  
}

Bài 2:

Code ban đầu:

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();  
 }  
 }  
}

Lỗi do Sonar phát hiện:

| **STT** | **Nội dung lỗi** | **Giải thích** | **Phân loại** |
| --- | --- | --- | --- |
| 1 | Replace this use of System.out by a logger (dòng 9) | Không nên dùng System.out.println(). Nên dùng logger để có thể kiểm soát log tốt hơn. | Coding Style / Best Practice |
| 2 | Move this file to a named package (dòng 0) | Nên tổ chức code trong các package có tên rõ ràng. | Coding Style / Structure |

Code hoàn chỉnh:

package com.example.security;  
  
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();  
 }  
 }  
}

Bài 3:

Code ban đầu

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();  
 }  
 }  
}

Lỗi do Sonar phát hiện:

| **STT** | **Nội dung lỗi** | **Giải thích** | **Phân loại** |
| --- | --- | --- | --- |
| 1 | Replace this use of System.out by a logger (dòng 9) | Dùng logger thay cho System.out.println() | Coding Style |
| 2 | Move this file to a named package (dòng 0) | Đưa vào package | Coding Style |
| 3 | Use try-with-resources or close this "BufferedReader" in a "finally" clause (dòng 6) | Để tránh resource leak (rò rỉ tài nguyên), nên dùng try-with-resources hoặc đảm bảo đóng BufferedReader | Coding Style / Resource management |

**Giải thích ngắn gọn:**

* Việc không đóng BufferedReader có thể gây rò rỉ tài nguyên (memory leak, file handle leak).
* try-with-resources giúp tự động đóng tài nguyên khi kết thúc khối lệnh.
* Dùng Logger để dễ quản lý log.
* Đưa code vào package cho đúng chuẩn tổ chức dự án.

Code hoàn chỉnh:

package com.example.security;  
  
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());  
 }  
 }  
}

Bài 4:

Code ban đầu:

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);  
 }  
}  
  
Lỗi do Sonar phát hiện:

| **STT** | **Nội dung lỗi** | **Giải thích** | **Phân loại** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Replace this use of System.out by a logger (dòng 7) | Dùng logger thay cho System.out.println() | Coding Style |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | Move this file to a named package (dòng 0) | Đưa file vào package | Coding Style |

**Giải thích ngắn gọn**

* Mục tiêu của bài này là chỉ fix các vấn đề theo SonarQube yêu cầu (giữ nguyên logic, chưa fix SQL Injection).
* Sonar chưa phân tích phần SQL Injection vì đoạn code này chưa thực sự thực thi query lên database.
* Chúng ta chỉ cần thêm logger và package như các bài trước.

Code hoàn chỉnh:

package com.example.security;  
  
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*.info("Executing query: " + query);  
 }  
}

Bài 5:

Code ban đầu:

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

Lỗi do Sonar phát hiện:

| **STT** | **Nội dung lỗi** | **Giải thích** | **Phân loại** |
| --- | --- | --- | --- |
| 1 | Move this file to a named package | File đang nằm ở default package → nên đặt vào package có tên | Coding Style |

**Giải thích ngắn gọn**

* SonarQube lần này chỉ nhắc nhở về việc nên tổ chức file vào package.
* Nhưng trên thực tế, đoạn code còn có 1 lỗi biên dịch (compile error) vì Circle chưa override phương thức draw() của interface Drawable.

Code hoàn chỉnh:

package com.example.security;  
  
interface Drawable {  
 void draw();  
}  
  
class Circle implements Drawable {  
 @Override  
 public void draw() {  
 System.*out*.println("Drawing a circle");  
 }  
}

Bài 6:

Code ban đầu:

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

Lỗi do Sonar phát hiện:

| **STT** | **Nội dung lỗi** | **Giải thích** | **Phân loại** |
| --- | --- | --- | --- |
| 1 | Move constants defined in this interface to another class or enum | Không nên định nghĩa hằng số trong interface, hãy chuyển sang class hoặc enum | Design |
| 2 | Move this file to a named package | Nên đặt vào package có tên | Coding Style |

**Giải thích ngắn gọn**

* Trong Java, việc định nghĩa constant trong interface (còn gọi là *Constant Interface Anti-pattern*) là **bad practice**.
* Các hằng số nên được đặt vào class final hoặc enum để dễ quản lý.
* SonarQube khuyến cáo chuyển các constant sang class hoặc enum.

Code hoàn chỉnh:

package com.example.security;  
  
// Đưa constants vào class final  
public final class Constants {  
 public static final int *MAX\_USERS* = 100;  
  
 private Constants() {  
 // Constructor private để tránh tạo instance  
 }  
}  
  
public class InterfaceFieldModificationExample {  
 public static void main(String[] args) {  
 System.*out*.println("Max users: " + Constants.*MAX\_USERS*);  
 }  
}

Bài 7:

Code ban đầu:

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

Các lỗi do Sonar phát hiện:

| **STT** | **Nội dung lỗi** | **Giải thích** | **Phân loại** |
| --- | --- | --- | --- |
| 1 | Replace this use of System.out by a logger (2 lần) | Không nên dùng System.out, nên dùng logger | Code Smell |
| 2 | Rename this method name to match the regular expression ^[a-z][a-zA-Z0-9]\*$ | Phương thức Speak viết sai quy ước đặt tên (bắt đầu phải là chữ thường) | Code Convention |
| 3 | Rename method "Speak" to prevent misunderstanding with method "speak" | Do phương thức ghi đè nhưng khác nhau về chữ hoa/thường, gây lỗi logic | Bug |
| 4 | Move this file to a named package | Nên đặt vào package có tên | Coding Style |
|  |  |  |  |

**Giải thích vấn đề chính**

* Trong Java, tên phương thức phân biệt chữ hoa/thường nên Speak() **không override** speak().
* Do đó phương thức speak() trong lớp cha vẫn hoạt động, còn Speak() trong Dog là một phương thức hoàn toàn khác.
* SonarQube yêu cầu sửa lại để ghi đè chính xác, dùng annotation @Override để compiler kiểm tra.

Code hoàn chỉnh:

package com.example.override;  
  
import java.util.logging.Logger;  
  
class Animal {  
 private static final Logger *logger* = Logger.*getLogger*(Animal.class.getName());  
  
 void speak() {  
 *logger*.info("Animal speaks");  
 }  
}  
  
class Dog extends Animal {  
 private static final Logger *logger* = Logger.*getLogger*(Dog.class.getName());  
  
 @Override  
 void speak() {  
 *logger*.info("Dog barks");  
 }  
}

Bài 8:

Code ban đầu:

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");   
 }  
 }  
}

Lỗi do Sonar phát hiện:

| **STT** | **Nội dung lỗi** | **Giải thích** | **Loại** |
| --- | --- | --- | --- |
| 1 | Replace this use of System.out by a logger | Thay System.out bằng logger | Code Smell |
| 2 | Replace e with an unnamed pattern | Nếu không dùng e, có thể bỏ tên biến | Code Smell |
| 3 | Move this file to a named package | Đưa vào package có tên | Maintainability |
| 4 | A NullPointerException could be thrown; s is nullable here | Có thể kiểm tra null trước khi gọi s.length() | Bug |
| 5 | Catch generic Exception | Không nên bắt Exception chung chung, nên bắt chính xác NullPointerException | Bug |

Code hoàn chỉnh:

package com.hainam.exception;  
  
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 = "Hello SonarQube"; // Giả lập giá trị thực tế  
  
 try {  
 if (s != null) {  
 *logger*.info(() -> String.*format*("Length: %d", s.length()));  
 } else {  
 *logger*.warning("String is null");  
 }  
 } catch (NullPointerException ignored) {  
 *logger*.severe("NullPointerException occurred");  
 }  
 }  
}

Bài 9:

Code ban đầu:

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

Lỗi do Sonar phát hiện:

| **STT** | **Mô tả lỗi** | **Giải thích** |
| --- | --- | --- |
| 1 | Replace this use of System.out by a logger | Không dùng System.out, nên dùng Logger |
| 2 | Use isEmpty() to check whether a string is empty or not | Đang kiểm tra bằng length() > 0, nên dùng isEmpty() |
| 3 | Move this file to a named package | File chưa nằm trong package |
| 4 | "NullPointerException" could be thrown; "text" is nullable here | Biến text có thể null nên .length() sẽ ném ra NPE |

**Cách fix chuẩn SonarQube:**

1️⃣ Luôn kiểm tra null trước khi gọi phương thức trên object.  
2️⃣ Dùng logger thay cho System.out.  
3️⃣ Thêm package cho file.  
4️⃣ Thay length() > 0 bằng !text.isEmpty().

Code hoàn chỉnh:

package com.hainam.errorcodes;  
  
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 = null;  
  
 if (text != null && !text.isEmpty()) {  
 *logger*.info("Text is not empty");  
 } else {  
 *logger*.warning("Text is null or empty");  
 }  
 }  
}

Bài 10:

Code ban đầu:

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");   
 }  
 }  
}

Lỗi do Sonar phát hiện:

| **STT** | **Mô tả lỗi** | **Giải thích** |
| --- | --- | --- |
| 1 | Replace this use of System.out by a logger | Không dùng System.out, thay bằng Logger |
| 2 | Overcatching Exception | Đang bắt RuntimeException quá rộng, chỉ nên bắt ngoại lệ cụ thể ArrayIndexOutOfBoundsException |
| 3 | Move this file to a named package | File chưa nằm trong package |

Code hoàn chỉnh:

package com.hainam.errorcodes;  
  
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];  
 System.*out*.println(arr[10]);  
 } catch (ArrayIndexOutOfBoundsException e) {  
 *logger*.severe("Array index out of bounds: " + e.getMessage());  
 }  
 }  
}

Bài 11:

Code ban đầu:

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...");  
 }  
}

Lỗi do Sonar phát hiện:

| **STT** | **Vấn đề** | **Giải thích** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| 1 | Replace this use of System.out by a logger | Theo SonarQube khuyến cáo không dùng System.out. |

|  |  |  |
| --- | --- | --- |
| 2 | Tight Coupling | Report phụ thuộc trực tiếp vào Printer làm giảm tính mở rộng. |

|  |  |  |
| --- | --- | --- |
| 3 | Move this file to a named package | File chưa nằm trong package nào. |

Code hoàn chỉnh:

package com.hainam.errorcodes;  
  
import java.util.logging.Logger;  
  
interface Printer {  
 void print(String message);  
}  
  
class ConsolePrinter implements Printer {  
 private static final Logger *logger* = Logger.*getLogger*(ConsolePrinter.class.getName());  
  
 @Override  
 public void print(String message) {  
 *logger*.info(message);  
 }  
}  
  
class Report {  
 private Printer printer;  
  
 public Report(Printer printer) {  
 this.printer = printer;  
 }  
  
 void generate() {  
 printer.print("Generating report...");  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Printer printer = new ConsolePrinter();  
 Report report = new Report(printer);  
 report.generate();  
 }  
}

Bài 12:

Code ban đầu:

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

Lỗi do sonar phát hiện:

| **STT** | **Vấn đề** | **Giải thích** |
| --- | --- | --- |
| 1 | Dùng public field | Vi phạm nguyên tắc **Encapsulation** — dữ liệu có thể bị chỉnh sửa từ bên ngoài mà không kiểm soát được. |
| 2 | Replace this use of System.out by a logger | SonarQube khuyến cáo. |
| 3 | Move this file to a named package | Chưa có package rõ ràng. |

Code hoàn chỉnh:

package com.hainam.errorcodes;  
  
import java.util.logging.Logger;  
  
public class User {  
 private static final Logger *logger* = Logger.*getLogger*(User.class.getName());  
  
 private String name;  
 private int age;  
  
 public User(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
  
 public void display() {  
 *logger*.info("Name: " + name + ", Age: " + age);  
 }  
}

Bài 13:

Code ban đầu:

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*());  
 }  
}

Lỗi do Sonar phát hiện:

| **STT** | **Vấn đề** | **Giải thích** |
| --- | --- | --- |
| 1 | System.out.println("This will never execute"); | Không bao giờ được thực hiện vì đứng sau return. |
| 2 | Replace System.out by a logger | Theo SonarQube khuyến nghị. |
| 3 | Move this file to a named package | Chưa có package rõ ràng. |

Code hoàn chỉnh:

package com.hainam.errorcodes;  
  
import java.util.logging.Logger;  
  
public class UnreachableCodeExample {  
 private static final Logger *logger* = Logger.*getLogger*(UnreachableCodeExample.class.getName());  
  
 public static int getNumber() {  
 return 42;  
 }  
  
 public static void main(String[] args) {  
 *logger*.info(String.*valueOf*(*getNumber*()));  
 }  
}

Bài 14:

Code ban đầu:

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

Lỗi do Sonar phát hiện:

| **STT** | **Vấn đề** | **Giải thích** |
| --- | --- | --- |
| 1 | Square chưa implement đầy đủ phương thức resize() từ interface Shape. | Vi phạm nguyên tắc Interface Segregation. |
| 2 | Replace this use of System.out by a logger. | Theo SonarQube khuyến nghị. |
| 3 | Move this file to a named package. | Chưa có package. |

Code hoàn chỉnh:

package com.hainam.errorcodes;  
  
import java.util.logging.Logger;  
  
interface Shape {  
 void draw();  
 void resize();  
}  
  
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");  
 }  
}

Bài 15:

Code ban đầu:

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

Lỗi do Sonar phát hiện:

| **STT** | **Vấn đề** | **Giải thích** |
| --- | --- | --- |
| 1 | Tên interface loginhandler không đúng chuẩn Java naming (nên viết hoa: LoginHandler) | Vi phạm Java Naming Convention |
| 2 | Thiếu kiểu trả về của phương thức login | Java yêu cầu khai báo kiểu trả về rõ ràng |
| 3 | Sai kiểu dữ liệu string, trong Java phải là String | Java case-sensitive |
| 4 | Cú pháp phương thức interface thiếu public và {} | Trong interface, chỉ khai báo phương thức, không cài đặt |

Code hoàn chỉnh:

package com.hainam.errorcodes;  
  
public interface LoginHandler {  
 boolean login(String username, String password);  
}