


TRƯỜNG ĐẠI HỌC TƯ THỰC QUỐC TẾ SÀI GÒN KHOA KỸ THUẬT & KHOA HỌC MÁY TÍNH PHÁT TRIỂN, VẬN HÀNH VÀ BẢO TRÌ PHẦN MỀM	<b>Lab 2</b> <b>Xét duyệt mã nguồn và đóng gói          phần mềm</b>	 The Saigon International University
--	---	---

## Mục tiêu

1. **Hiểu quy trình xét duyệt mã nguồn:** Sinh viên hiểu được quy trình xét duyệt mã và tầm quan trọng của việc đảm bảo chất lượng mã trong phát triển phần mềm.
2. **Thành thạo sử dụng công cụ SonarQube:** Sinh viên sử dụng công cụ SonarQube để phân tích chất lượng mã nguồn và phát hiện các lỗi phổ biến (bug, code smell, lỗ hổng bảo mật).
3. **Đóng gói và phát hành phần mềm:** Sinh viên học cách đóng gói phần mềm Java thành các định dạng như JAR và triển khai lên hosting nếu cần.
4. **Nâng cao kỹ năng phát hiện và tối ưu lỗi mã nguồn:** Sinh viên tự nhận diện lỗi, trả lời câu hỏi để hiểu rõ vấn đề và đề xuất giải pháp cải thiện mã.

## Phần 1: Tutorial - Cài đặt và Cấu hình SonarQube và SonarScanner

### 1. Cài đặt SonarQube với Docker

1. **Yêu cầu:** Máy đã cài đặt Docker Desktop.
2. **Chạy SonarQube Server:**

```
docker pull sonarqube
docker run -d --name sonarqube -p 9000:9000 sonarqube
```

3. **Truy cập:** Mở trình duyệt và truy cập <http://localhost:9000> để đảm bảo SonarQube server hoạt động.

### 2. Cài đặt SonarScanner

- **Tải SonarScanner** từ [SonarScanner Download Page](#).
- **Cấu hình SonarScanner:**
  - Mở file sonar-scanner.properties.
  - Thêm thông tin kết nối SonarQube server:

```
sonar.host.url=http://localhost:9000
sonar.login=your_token # Token sẽ được sinh từ SonarQube
```

### 3. Tài liệu Tham khảo và Các Nguồn Tutorial Hỗ trợ

- **SonarQube Documentation:** [SonarQube Official Guide](#)
- **SonarScanner Documentation:** [SonarScanner Setup and Configuration](#)
- **Cài đặt SonarQube với Docker:** [SonarQube Docker Setup Guide](#)

### Phần 2: Phân tích các đoạn mã mẫu

Dưới đây là các đoạn mã chứa các lỗi phổ biến về logic, hiệu năng, bảo mật, và **code smell**. Sinh viên sử dụng SonarQube để phân tích từng đoạn mã, tự phát hiện lỗi và trả lời các câu hỏi để hiểu rõ vấn đề. Sau đó, sinh viên đề xuất giải pháp cải thiện mã.

#### Đoạn mã 1:

```
public class Calculator {  
  
    public int add(int a, int b) {  
        return a - b;  
    }  
  
    public int subtract(int a, int b) {  
        return a + b;  
    }  
  
    public boolean isPositive(int number) {  
        if (number <= 0) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

#### Câu hỏi:

1. Các phép toán trong add() và subtract() có chính xác không?

2. `isPositive()` có trả về đúng khi số là dương không?

**Đoạn mã 2:**

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class FileUtil {

    public String readFile(String filePath) {
        BufferedReader reader = null;
        StringBuilder content = new StringBuilder();

        try {
            reader = new BufferedReader(new
FileReader(filePath));
            String line;
            while ((line = reader.readLine()) != null) {
                content.append(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading file");
        }

        return content.toString();
    }
}
```

**Câu hỏi:**

1. `readFile()` có đảm bảo `BufferedReader` được đóng đúng cách không?
2. Có giải pháp nào để đảm bảo tài nguyên luôn được giải phóng

### Đoạn mã 3:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class UserDatabase {

    private String url = "jdbc:mysql://localhost:3306/users";
    private String user = "root";
    private String password = "password";

    public void getUserData(String username) {
        try (Connection conn = DriverManager.getConnection(url,
            user, password);
            Statement stmt = conn.createStatement()) {

            String query = "SELECT * FROM users WHERE username = "
                + username + "'";

            ResultSet rs = stmt.executeQuery(query);
            while (rs.next()) {
                System.out.println("User: " +
                    rs.getString("username"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

**Câu hỏi:**

1. getUserData() có bảo mật tốt không? Có rủi ro nào trong câu truy vấn SQL không?
2. Cách nào để ngăn chặn SQL Injection trong mã này?

**Đoạn mã 4:**

```
public class DivisionCalculator {  
  
    public int divide(int a, int b) {  
        return a / b;  
    }  
}
```

**Câu hỏi:**

1. Điều gì sẽ xảy ra nếu  $b = 0$ ?
2. Có cách nào để cải thiện mã nhằm tránh lỗi tiềm ẩn này không?

**Đoạn mã 5:**

```
public class OrderProcessor {  
  
    public void processOrder() {  
        System.out.println("Start processing order...");  
        System.out.println("Checking order validity...");  
        System.out.println("Validating customer details...");  
        System.out.println("Checking inventory...");  
        System.out.println("Reserving items...");  
        System.out.println("Calculating order total...");  
        System.out.println("Applying discounts...");  
        System.out.println("Processing payment...");  
        System.out.println("Confirming payment status...");  
        System.out.println("Generating invoice...");  
        System.out.println("Packing items...");  
    }  
}
```

```
        System.out.println("Arranging shipping...");
        System.out.println("Updating order status...");
        System.out.println("Sending confirmation email...");
        System.out.println("Logging transaction...");
        System.out.println("Updating customer records...");
        System.out.println("End processing order.");
    }
}
```

### Câu hỏi:

1. processOrder() có quá nhiều chức năng không?
2. Có thể chia thành các phương thức con không? Nếu có, hãy đề xuất các phương thức phù hợp.

### Đoạn mã 6:

```
public class DataProcessor {

    public void processData1() {
        System.out.println("Connecting to database...");
        System.out.println("Fetching data...");
        System.out.println("Processing data...");
        System.out.println("Closing connection...");
    }

    public void processData2() {
        System.out.println("Connecting to database...");
        System.out.println("Fetching data...");
        System.out.println("Processing data...");
        System.out.println("Closing connection...");
    }
}
```

**Câu hỏi:**

1. processData1() và processData2() có gì giống nhau không?
2. Có cách nào loại bỏ trùng lặp mã trong trường hợp này?

**Đoạn mã 7:**

```
public class OrderService {

    private String orderStatus = "Pending";

    public void createOrder() {
        System.out.println("Creating order...");
        System.out.println("Order created with status: " +
orderStatus);
    }

    public void cancelOrder() {
        String reason = "Customer Request"; // Biến này không
được sử dụng
        System.out.println("Cancelling order...");
        System.out.println("Order cancelled with status: " +
orderStatus);
    }

    public void confirmOrder() {
        System.out.println("Confirming order...");
        System.out.println("Order confirmed with status: " +
orderStatus);
    }
}
```

**Câu hỏi:**

1. Có biến nào không sử dụng trong đoạn mã này không?
2. Có cách nào giảm thiểu sự trùng lặp mã trong các phương thức `createOrder()`, `cancelOrder()`, và `confirmOrder()` không?

#### **Đoạn mã 8:**

```
import java.util.ArrayList;
import java.util.List;

public class UserManager {

    private List<String> users = new ArrayList<>();

    public void addUser(String username) {
        users.add(username);
        System.out.println("User added: " + username);
        logUserAdded(username);
    }

    public void logUserAdded(String username) {
        // Ghi log vào file (đơn giản hóa)
        System.out.println("Log: User added - " + username);
    }

    public List<String> getAllUsers() {
        return users;
    }
}
```

#### **Câu hỏi:**

1. Phương thức `addUser()` có đảm nhận quá nhiều trách nhiệm không?
2. Làm thế nào để tách chức năng ghi log ra khỏi `UserManager` để tuân thủ nguyên tắc Single Responsibility?



**Đoạn mã 9:**

```
public class MathOperations {

    public int findMax(int[] numbers) {
        if (numbers.length == 0) {
            return 0; // Giá trị trả về không hợp lý nếu mảng
trống
        }
        int max = numbers[0];
        for (int number : numbers) {
            if (number > max) {
                max = number;
            }
        }
        return max;
    }
}
```

**Câu hỏi:**

1. Giá trị trả về của findMax() có hợp lý không nếu mảng truyền vào là rỗng?
2. Có cách nào khác để xử lý trường hợp mảng rỗng?

**Đoạn mã 10:**

```
public class ShoppingCart {

    private List<String> items = new ArrayList<>();
    private double totalAmount;

    public void checkout() {
        System.out.println("Starting checkout...");
    }
}
```

```
        for (String item : items) {
            System.out.println("Processing item: " + item);
            if (item.equals("Discount")) {
                totalAmount -= 5;
            } else {
                totalAmount += 10;
            }
        }

        if (totalAmount < 0) {
            totalAmount = 0;
        }

        System.out.println("Applying taxes...");
        totalAmount *= 1.08;
        System.out.println("Final amount: " + totalAmount);
        System.out.println("Checkout complete.");
    }
}
```

### Câu hỏi:

1. Phương thức checkout() có quá nhiều chức năng không? Có thể tách ra thành các phương thức nhỏ hơn không?
2. Có cách nào để xử lý các phần như "áp dụng giảm giá" và "tính thuế" trong các phương thức riêng?

### Đoạn mã 11:

```
public class PaymentProcessor {
```

```

    private double amt; // Tên biến không rõ ràng

    private double taxRate = 0.1; // Biến không cần thiết khi
thuế là cố định

    public double processPayment(double amount) {
        double tempAmount = amount; // Biến này không cần thiết
        double finalAmount = tempAmount + (tempAmount *
taxRate);
        return finalAmount;
    }
}

```

### Câu hỏi:

1. Biến amt và tempAmount có cần thiết không? Có cách nào tối ưu hóa mã để không sử dụng biến này?
2. Tên biến amt có ý nghĩa không? Có thể đổi tên biến để tăng tính rõ ràng không?

### Đoạn mã 12:

```

public class ReportGenerator {

    public void generateReport() {
        Report report = new Report();
        report.create();
        System.out.println("Report created");

        Report report2 = new Report(); // Đối tượng không cần
thiết
        report2.create();
        System.out.println("Report created again");
    }
}

```

```
        if (report.isReady()) {  
            System.out.println("Report is ready to download");  
        }  
    }  
}
```

### Câu hỏi:

1. Việc khởi tạo Report lần thứ hai có cần thiết không?
2. Có phần nào trùng lặp mà có thể gộp lại để tối ưu không?

### Hướng dẫn Sinh viên Phân tích và Báo cáo

Sinh viên cần thực hiện các bước sau với SonarQube để phát hiện lỗi và tối ưu mã:

1. **Chạy SonarScanner:** Phân tích từng đoạn mã với SonarQube và ghi lại các lỗi phát hiện được.
2. **Ghi nhận lỗi và câu hỏi tự luận:**
  - Ghi lại các lỗi mà SonarQube phát hiện (biến không cần thiết, mã trùng lặp, hàm không trả về hợp lý).
  - Trả lời câu hỏi kèm theo để xác định lỗi và hiểu rõ nguyên nhân.
3. **Đề xuất cách cải thiện:**
  - Đưa ra cách sửa lỗi và tối ưu mã (như xóa biến không dùng, tách phương thức).
4. **Báo cáo:**
  - Nộp báo cáo tổng hợp gồm danh sách lỗi phát hiện, câu trả lời và các đề xuất cải thiện mã cho từng đoạn.

### Yêu cầu Bài nộp

- **Báo cáo phân tích lỗi:** Ghi lại các lỗi code smell, cách phân loại và đề xuất cải thiện mã.
- **Báo cáo từ SonarQube:** Xuất file báo cáo từ SonarQube và nộp lại.