**ĐOẠN CODE VI PHẠM NGUYÊN TẮC CODE KISS/DRY/YAGNI**

* Cho đoạn code vi phạm nguyên tắc code bên dưới, nhóm cùng phân tích vấn đề của code vi phạm các nguyên tắc KISS/DRY/YAGNI. Hãy thảo luận cách refactor sau đó thực hiện refactor code đó.
* **Git/GitHub:** Tạo branch mới trên GitHub để thực hiện refactor (ví dụ: feature/ refactor-code). Thực hiện các commit rõ ràng trong quá trình refactor.
* Viết tài liệu hướng dẫn các bước đã thực hiện để refactor code gồm:
* Mô tả mục tiêu của đoạn code
* Chỉ ra các vấn đề của đoạn code ban đầu: vi phạm nguyên tắc nào?.
* Giải thích các nguyên tắc đã áp dụng.
* Trình bày các bước refactor cụ thể (ví dụ: "Bước 1: Tách hàm ghi log", "Bước 2: Tách hàm gửi email", "Bước 3: Tạo các hàm chuyên biệt cho CRUD").
* Giải thích lý do cho từng thay đổi.
* So sánh đoạn code trước và sau refactor để thấy sự cải thiện.
* Tài liệu các bước để đưa lên GitHub và các nhánh mà nhóm đã tạo vào file Word (gộp chung với tài liệu Refctor code).

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.time.format.DateTimeParseException;

import java.util.ArrayList;

import java.util.List;

import java.util.UUID;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import org.json.simple.parser.ParseException;

public class PersonalTaskManagerViolations {

    private static final String DB\_FILE\_PATH = "tasks\_database.json";

    private static final DateTimeFormatter DATE\_FORMATTER = DateTimeFormatter.ofPattern("yyyy-MM-dd");

    // Phương thức trợ giúp để tải dữ liệu (sẽ được gọi lặp lại)

    private static JSONArray loadTasksFromDb() {

        JSONParser parser = new JSONParser();

        try (FileReader reader = new FileReader(DB\_FILE\_PATH)) {

            Object obj = parser.parse(reader);

            if (obj instanceof JSONArray) {

                return (JSONArray) obj;

            }

        } catch (IOException | ParseException e) {

            System.err.println("Lỗi khi đọc file database: " + e.getMessage());

        }

        return new JSONArray();

    }

    // Phương thức trợ giúp để lưu dữ liệu

    private static void saveTasksToDb(JSONArray tasksData) {

        try (FileWriter file = new FileWriter(DB\_FILE\_PATH)) {

            file.write(tasksData.toJSONString());

            file.flush();

        } catch (IOException e) {

            System.err.println("Lỗi khi ghi vào file database: " + e.getMessage());

        }

    }

    /\*\*

     \* Chức năng thêm nhiệm vụ mới

     \*

     \* @param title Tiêu đề nhiệm vụ.

     \* @param description Mô tả nhiệm vụ.

     \* @param dueDateStr Ngày đến hạn (định dạng YYYY-MM-DD).

     \* @param priorityLevel Mức độ ưu tiên ("Thấp", "Trung bình", "Cao").

     \* @param isRecurring Boolean có phải là nhiệm vụ lặp lại không.

     \* @return JSONObject của nhiệm vụ đã thêm, hoặc null nếu có lỗi.

     \*/

    public JSONObject addNewTaskWithViolations(String title, String description,

                                                String dueDateStr, String priorityLevel,

                                                boolean isRecurring) {

        if (title == null || title.trim().isEmpty()) {

            System.out.println("Lỗi: Tiêu đề không được để trống.");

            return null;

        }

        if (dueDateStr == null || dueDateStr.trim().isEmpty()) {

            System.out.println("Lỗi: Ngày đến hạn không được để trống.");

            return null;

        }

        LocalDate dueDate;

        try {

            dueDate = LocalDate.parse(dueDateStr, DATE\_FORMATTER);

        } catch (DateTimeParseException e) {

            System.out.println("Lỗi: Ngày đến hạn không hợp lệ. Vui lòng sử dụng định dạng YYYY-MM-DD.");

            return null;

        }

        String[] validPriorities = {"Thấp", "Trung bình", "Cao"};

        boolean isValidPriority = false;

        for (String validP : validPriorities) {

            if (validP.equals(priorityLevel)) {

                isValidPriority = true;

                break;

            }

        }

        if (!isValidPriority) {

            System.out.println("Lỗi: Mức độ ưu tiên không hợp lệ. Vui lòng chọn từ: Thấp, Trung bình, Cao.");

            return null;

        }

        // Tải dữ liệu

        JSONArray tasks = loadTasksFromDb();

        // Kiểm tra trùng lặp

        for (Object obj : tasks) {

            JSONObject existingTask = (JSONObject) obj;

            if (existingTask.get("title").toString().equalsIgnoreCase(title) &&

                existingTask.get("due\_date").toString().equals(dueDate.format(DATE\_FORMATTER))) {

                System.out.println(String.format("Lỗi: Nhiệm vụ '%s' đã tồn tại với cùng ngày đến hạn.", title));

                return null;

            }

        }

        String taskId = UUID.randomUUID().toString(); // YAGNI: Có thể dùng số nguyên tăng dần đơn giản hơn.

        JSONObject newTask = new JSONObject();

        newTask.put("id", taskId);

        newTask.put("title", title);

        newTask.put("description", description);

        newTask.put("due\_date", dueDate.format(DATE\_FORMATTER));

        newTask.put("priority", priorityLevel);

        newTask.put("status", "Chưa hoàn thành");

        newTask.put("created\_at", LocalDateTime.now().format(DateTimeFormatter.ISO\_DATE\_TIME));

        newTask.put("last\_updated\_at", LocalDateTime.now().format(DateTimeFormatter.ISO\_DATE\_TIME));

        newTask.put("is\_recurring", isRecurring); // YAGNI: Thêm thuộc tính này dù chưa có chức năng xử lý nhiệm vụ lặp lại

        if (isRecurring) {

            newTask.put("recurrence\_pattern", "Chưa xác định");

        }

        tasks.add(newTask);

        // Lưu dữ liệu

        saveTasksToDb(tasks);

        System.out.println(String.format("Đã thêm nhiệm vụ mới thành công với ID: %s", taskId));

        return newTask;

    }

    public static void main(String[] args) {

        PersonalTaskManagerViolations manager = new PersonalTaskManagerViolations();

        System.out.println("\nThêm nhiệm vụ hợp lệ:");

        manager.addNewTaskWithViolations(

            "Mua sách",

            "Sách Công nghệ phần mềm.",

            "2025-07-20",

            "Cao",

            false

        );

        System.out.println("\nThêm nhiệm vụ trùng lặp (minh họa DRY - lặp lại code đọc/ghi DB và kiểm tra trùng):");

        manager.addNewTaskWithViolations(

            "Mua sách",

            "Sách Công nghệ phần mềm.",

            "2025-07-20",

            "Cao",

            false

        );

        System.out.println("\nThêm nhiệm vụ lặp lại (minh họa YAGNI - thêm tính năng không cần thiết ngay):");

        manager.addNewTaskWithViolations(

            "Tập thể dục",

            "Tập gym 1 tiếng.",

            "2025-07-21",

            "Trung bình",

            true

        );

        System.out.println("\nThêm nhiệm vụ với tiêu đề rỗng:");

        manager.addNewTaskWithViolations(

            "",

            "Nhiệm vụ không có tiêu đề.",

            "2025-07-22",

            "Thấp",

            false

        );

    }

}

## 🔰 **1. Mục tiêu của đoạn code**

Đoạn mã PersonalTaskManagerViolations cung cấp chức năng:

* Thêm một nhiệm vụ cá nhân vào cơ sở dữ liệu dạng JSON (tasks\_database.json).
* Kiểm tra dữ liệu đầu vào (tiêu đề, ngày đến hạn, mức ưu tiên).
* Phát hiện nhiệm vụ trùng lặp (cùng tiêu đề và ngày đến hạn).
* Ghi nhận thông tin như mức độ ưu tiên, trạng thái, thời điểm tạo, và xử lý nhiệm vụ lặp lại.

## ⚠️ **2. Các vấn đề và nguyên tắc thiết kế bị vi phạm**

| **Vấn đề** | **Nguyên tắc bị vi phạm** | **Giải thích** |
| --- | --- | --- |
| Hàm addNewTaskWithViolations() quá dài | **SRP (Single Responsibility Principle)** | Làm quá nhiều việc: kiểm tra, xử lý logic, lưu file |
| Lặp lại đoạn đọc/ghi file | **DRY (Don't Repeat Yourself)** | Cùng logic xử lý nhưng lặp lại ở nhiều nơi |
| Dùng UUID, thêm trường is\_recurring mà chưa dùng | **YAGNI (You Aren’t Gonna Need It)** | Thêm tính năng chưa cần thiết |
| Mảng mức độ ưu tiên khai báo cứng | **Avoid Hardcoded Values** | Khó mở rộng, không tái sử dụng |
| Logic kiểm tra trùng lặp viết tay | **KISS (Keep It Simple, Stupid)** | Cần trừu tượng hóa cho dễ đọc và bảo trì |

## 🧱 **3. Nguyên tắc thiết kế áp dụng khi refactor**

* **SRP**: Mỗi hàm chỉ có một trách nhiệm rõ ràng.
* **DRY**: Tách các đoạn mã lặp lại thành phương thức tái sử dụng.
* **KISS**: Giữ code ngắn gọn, dễ đọc, dễ hiểu.
* **Encapsulation**: Gói dữ liệu và hành vi vào các đối tượng/phương thức liên quan.
* **Modularization**: Phân chia logic thành các khối chuyên biệt.

## 🔧 **4. Các bước refactor cụ thể**

### ✅ **Bước 1: Tạo class** Task **để đại diện nhiệm vụ**

java

Sao chépChỉnh sửa

public class Task {

private String id;

private String title;

private String description;

private LocalDate dueDate;

private String priority;

private String status;

private LocalDateTime createdAt;

private LocalDateTime lastUpdatedAt;

// Các constructor, getter/setter, toJSON() sẽ hỗ trợ việc serialize ra JSON

}

**💡Lý do:** Giúp mã rõ ràng, dễ mở rộng và tránh thao tác trực tiếp với JSONObject.

### ✅ **Bước 2: Tách logic kiểm tra đầu vào thành phương thức riêng**

java

Sao chépChỉnh sửa

private boolean isValidInput(String title, String dueDateStr, String priorityLevel) { ... }

**💡Lý do:** Tránh nhồi nhét logic kiểm tra vào hàm chính.

### ✅ **Bước 3: Tách logic kiểm tra trùng nhiệm vụ**

java

Sao chépChỉnh sửa

private boolean isDuplicateTask(JSONArray tasks, String title, LocalDate dueDate) { ... }

**💡Lý do:** Đơn giản hóa hàm chính và tách trách nhiệm rõ ràng.

### ✅ **Bước 4: Tách phương thức đọc/ghi JSON**

java

Sao chépChỉnh sửa

private JSONArray loadTasksFromFile() { ... }

private void saveTasksToFile(JSONArray tasks) { ... }

**💡Lý do:** Tránh lặp lại mã đọc/ghi file.

### ✅ **Bước 5: Áp dụng Enum cho mức độ ưu tiên**

java

Sao chépChỉnh sửa

public enum Priority {

THAP, TRUNG\_BINH, CAO;

public static boolean isValid(String input) { ... }

}

**💡Lý do:** Dễ kiểm soát và mở rộng thay vì dùng mảng chuỗi cứng.

## 📌 **5. Giải thích lý do cho từng thay đổi**

| **Thay đổi** | **Lý do chính** |
| --- | --- |
| Tạo class Task | Gói dữ liệu, dễ serialize/deserialize |
| Tách hàm kiểm tra input | SRP + dễ kiểm thử |
| Enum Priority | Tránh lỗi chính tả, dễ mở rộng |
| Hàm kiểm tra trùng lặp | KISS + SRP |
| Tách logic JSON | DRY, dễ sửa đổi backend sau này |

## 🔍 **6. So sánh trước và sau refactor**

| **Tiêu chí** | **Trước refactor** | **Sau refactor** |
| --- | --- | --- |
| **Độ dài hàm chính** | Dài (~80 dòng) | Ngắn gọn (~10-15 dòng) |
| **Mức độ phân tách trách nhiệm** | Không rõ ràng | Rõ ràng theo từng hàm |
| **Tính mở rộng** | Khó thêm tính năng (ví dụ: cập nhật nhiệm vụ) | Dễ mở rộng qua class Task |
| **Tái sử dụng mã** | Không có | Có (validate, save/load, duplicate check) |
| **Tính dễ đọc và bảo trì** | Thấp | Cao |