**BÁO CÁO BÀI TẬP LỚN**

**Tên Đề Tài: Xây Dựng Ứng Dụng Học Từ Vựng Tiếng Anh**

**1. Giới thiệu và Phân tích Bài toán**

**1.1. Giới thiệu bài toán**

Ngày nay, việc học ngoại ngữ, đặc biệt là tiếng Anh, đã trở thành một nhu cầu thiết yếu. Một trong những yếu tố cốt lõi của việc học ngoại ngữ là xây dựng vốn từ vựng. Ứng dụng được xây dựng là một công cụ dòng lệnh (console application) đơn giản nhằm giúp người dùng quản lý và học các từ vựng tiếng Anh.

Vấn đề cốt lõi mà ứng dụng giải quyết là cung cấp một nền tảng linh hoạt để người dùng có thể thêm từ mới và lựa chọn các phương pháp học khác nhau (như flashcard hoặc trắc nghiệm) để ôn luyện. Bài toán đặt ra yêu cầu hệ thống phải dễ dàng bảo trì và mở rộng—ví dụ, thêm các phương pháp học mới trong tương lai mà không cần sửa đổi nhiều mã nguồn hiện có.

**1.2. Yêu cầu Chức năng và Phi chức năng**

**Yêu cầu Chức năng (Functional Requirements):**

* **F1:** Hệ thống phải cho phép người dùng thêm một từ vựng mới, bao gồm: từ (term), nghĩa (meaning), và ví dụ (example).
* **F2:** Hệ thống phải kiểm tra và ngăn không cho người dùng thêm các từ đã tồn tại.
* **F3:** Hệ thống phải cung cấp chức năng học từ vựng theo phương pháp **Flashcard**: hiển thị từ, chờ người dùng ra hiệu lệnh, sau đó hiển thị nghĩa.
* **F4:** Hệ thống phải cung cấp chức năng học từ vựng theo phương pháp **Trắc nghiệm (Multiple Choice Quiz)**: với mỗi từ, hệ thống đưa ra một câu hỏi và 4 đáp án (1 đúng, 3 sai) để người dùng lựa chọn.
* **F5:** Hệ thống phải thông báo cho người dùng mỗi khi một từ mới được thêm vào kho từ vựng.
* **F6:** Người dùng có thể thoát khỏi ứng dụng.

**Yêu cầu Phi chức năng (Non-functional Requirements):**

* **Tính mở rộng (Extensibility):** Hệ thống phải dễ dàng thêm các phương pháp học mới (ví dụ: điền vào chỗ trống, nghe và đoán từ) mà không ảnh hưởng đến các chức năng hiện có.
* **Tính bảo trì (Maintainability):** Mã nguồn cần được tổ chức rõ ràng, dễ hiểu, và dễ sửa đổi. Các thành phần logic phải được phân tách rạch ròi.
* **Tính linh hoạt (Flexibility):** Người dùng có thể tự do lựa chọn phương pháp học tại thời điểm chạy (runtime).
* **Tính tái sử dụng (Reusability):** Các thành phần như logic quản lý từ vựng, các phương pháp học nên được thiết kế để có thể tái sử dụng.

**1.3. Biểu đồ Use Case và Mô tả**

**Các Use Case chính:**

* **Use Case 1: Học theo Flashcard**
  + **Tác nhân:** Người dùng
  + **Mô tả:** Người dùng chọn chức năng "Học theo Flashcard". Hệ thống sẽ lần lượt duyệt qua danh sách từ vựng. Với mỗi từ, hệ thống hiển thị từ đó và chờ người dùng nhấn Enter. Sau khi người dùng nhấn Enter, hệ thống sẽ hiển thị nghĩa của từ.
* **Use Case 2: Học theo Quiz ABCD**
  + **Tác nhân:** Người dùng
  + **Mô tả:** Người dùng chọn chức năng "Học theo Quiz". Hệ thống sẽ lần lượt duyệt qua danh sách từ vựng. Với mỗi từ, hệ thống hiển thị từ đó kèm theo 4 lựa chọn về nghĩa. Người dùng nhập đáp án (A, B, C, hoặc D). Hệ thống kiểm tra và thông báo kết quả "Đúng" hoặc "Sai" và hiển thị đáp án đúng.
* **Use Case 3: Thêm từ mới**
  + **Tác nhân:** Người dùng
  + **Mô tả:** Người dùng chọn chức năng "Thêm từ mới". Hệ thống yêu cầu nhập từ, nghĩa, và ví dụ. Hệ thống kiểm tra xem từ đã tồn tại chưa. Nếu chưa, từ mới sẽ được lưu lại và hệ thống hiển thị thông báo thành công cùng với tổng số từ vựng hiện có. Nếu đã tồn tại, hệ thống báo lỗi.

**2. Thiết kế Hệ thống**

**2.1. Mô hình Domain (Domain Model)**

Mô hình domain tập trung vào các thực thể nghiệp vụ cốt lõi của bài toán.

* **Word:** Đại diện cho một đơn vị từ vựng.
* **VocabularyManager:** Chịu trách nhiệm quản lý toàn bộ kho từ vựng.

**2.2. Nhận diện Vấn đề Thiết kế và Áp dụng Mẫu Thiết kế**

1. **Vấn đề 1: Quản lý trạng thái toàn cục:** Làm thế nào để đảm bảo rằng chỉ có **một và chỉ một** đối tượng quản lý kho từ vựng trong toàn bộ ứng dụng, tránh việc dữ liệu bị phân mảnh và không nhất quán?

**Giải pháp: Áp dụng Mẫu Singleton** cho lớp VocabularyManager.

1. **Vấn đề 2: Thiếu linh hoạt trong việc mở rộng phương pháp học:** Nếu viết logic học flashcard và quiz trực tiếp trong lớp Main, khi muốn thêm một phương pháp học mới (ví dụ: TypingTestStrategy), chúng ta sẽ phải sửa đổi lớp Main bằng cách thêm nhiều câu lệnh if-else hoặc switch-case. Điều này vi phạm **nguyên lý Mở/Đóng (OCP)**.

**Giải pháp: Áp dụng Mẫu Strategy**. Tách các thuật toán/phương pháp học thành các lớp riêng biệt, tuân theo một giao diện chung (LearningStrategy).

1. **Vấn đề 3: Phụ thuộc chặt chẽ khi thông báo thay đổi:** Lớp VocabularyManager cần thông báo cho các thành phần khác (ví dụ: bộ đếm tiến độ LearningProgress) mỗi khi có từ mới được thêm vào. Nếu VocabularyManager gọi trực tiếp một phương thức của LearningProgress, nó sẽ tạo ra một sự phụ thuộc chặt chẽ, khiến VocabularyManager khó có thể thông báo cho các đối tượng khác trong tương lai.

**Giải pháp: Áp dụng Mẫu Observer**. VocabularyManager (Subject) chỉ cần biết về một danh sách các Observer trừu tượng và thông báo cho chúng khi có sự kiện xảy ra, không cần biết chúng là ai hay làm gì.

**2.3. Áp dụng Nguyên lý SOLID**

* **SRP (Single Responsibility Principle):** Mỗi lớp có một trách nhiệm duy nhất.
  + Word: Chỉ chứa dữ liệu của một từ.
  + VocabularyManager: Chỉ quản lý danh sách từ và các observer.
  + FlashcardStrategy, MultipleChoiceQuizStrategy: Mỗi lớp chỉ chịu trách nhiệm cho một thuật toán học cụ thể.
  + LearningProgress: Chỉ chịu trách nhiệm theo dõi và hiển thị tiến độ.
* **OCP (Open/Closed Principle):** Hệ thống mở cho việc mở rộng nhưng đóng cho việc sửa đổi.
  + Nhờ có mẫu **Strategy**, chúng ta có thể thêm một phương pháp học mới bằng cách tạo một lớp mới implement LearningStrategy mà không cần sửa đổi mã nguồn của lớp Main hay VocabularyManager.
* **LSP (Liskov Substitution Principle):** Các đối tượng của lớp con có thể thay thế cho các đối tượng của lớp cha mà không làm thay đổi tính đúng đắn của chương trình.
  + Trong lớp Main, một đối tượng LearningStrategy có thể là FlashcardStrategy hoặc MultipleChoiceQuizStrategy. Mã nguồn vẫn hoạt động đúng vì cả hai đều tuân thủ hợp đồng của LearningStrategy.
* **ISP (Interface Segregation Principle):** Các interface được thiết kế nhỏ và tập trung.
  + LearningStrategy và ProgressObserver là hai interface riêng biệt cho hai mục đích khác nhau, tránh việc một lớp phải implement những phương thức mà nó không dùng đến.
* **DIP (Dependency Inversion Principle):** Các module cấp cao không nên phụ thuộc vào các module cấp thấp. Cả hai nên phụ thuộc vào sự trừu tượng.
  + Main (cấp cao) phụ thuộc vào interface LearningStrategy (trừu tượng), chứ không phụ thuộc vào FlashcardStrategy hay MultipleChoiceQuizStrategy (cấp thấp).
  + VocabularyManager (cấp cao) phụ thuộc vào interface ProgressObserver (trừu tượng), chứ không phụ thuộc vào lớp LearningProgress (cấp thấp).

**2.4. Chi tiết các Mẫu Thiết kế đã áp dụng**

**a. Mẫu Singleton (Creational)**

* **Lý do chọn:** Đảm bảo chỉ có một thực thể duy nhất của VocabularyManager tồn tại, cung cấp một điểm truy cập toàn cục và duy nhất đến kho từ vựng.
* **Biểu đồ lớp:**
* **Giải quyết vấn đề:** Mẫu này giải quyết vấn đề quản lý trạng thái toàn cục, đảm bảo dữ liệu từ vựng là nhất quán trên toàn bộ ứng dụng.

**b. Mẫu Strategy (Behavioral)**

* **Lý do chọn:** Cho phép thay đổi thuật toán (phương pháp học) tại thời điểm chạy. Giúp hệ thống dễ dàng mở rộng bằng cách thêm các chiến lược mới mà không vi phạm nguyên lý OCP.
* **Biểu đồ lớp:**
* **Giải quyết vấn đề:** Thay vì dùng if-else trong Main, chúng ta đóng gói các thuật toán học vào các đối tượng riêng biệt. Main chỉ cần biết về interface LearningStrategy, làm cho code linh hoạt và dễ mở rộng hơn nhiều.

**c. Mẫu Observer (Behavioral)**

* **Lý do chọn:** Tạo ra một cơ chế đăng ký và thông báo linh hoạt. Khi trạng thái của VocabularyManager (Subject) thay đổi (thêm từ mới), tất cả các Observer (ví dụ: LearningProgress) đã đăng ký sẽ tự động được thông báo và cập nhật.
* **Biểu đồ lớp:**
* **Giải quyết vấn đề:** Mẫu này giúp loại bỏ sự phụ thuộc chặt chẽ giữa VocabularyManager và LearningProgress. VocabularyManager không cần biết chi tiết về LearningProgress, nó chỉ cần gửi thông báo. Ta có thể dễ dàng thêm các Observer khác (ví dụ: một LoggerObserver ghi log ra file) mà không cần sửa đổi VocabularyManager.

**2.5. Biểu đồ Tuần tự (Sequence Diagram)**

**Luồng chức năng "Thêm từ mới" (Minh họa Observer Pattern)**

**Luồng chức năng "Học theo Quiz" (Minh họa Strategy Pattern)**

**3. Triển khai (Coding) và Kiểm thử**

**3.1. Chất lượng Code**

Mã nguồn được tổ chức thành các file riêng biệt, mỗi file tương ứng với một lớp hoặc interface, tuân thủ nguyên lý SRP. Tên lớp, phương thức và biến rõ ràng, thể hiện đúng chức năng. Việc sử dụng các mẫu thiết kế giúp cấu trúc code trở nên mạch lạc và dễ hiểu.

**3.2. Các đoạn code minh họa**

**Triển khai Mẫu Singleton trong VocabularyManager.java**

public class VocabularyManager {

private static VocabularyManager instance;

//...

private VocabularyManager() {

// private constructor

}

public static VocabularyManager getInstance() {

if (instance == null) {

instance = new VocabularyManager();

}

return instance;

}

//...

}

**Triển khai Mẫu Strategy (Interface và các lớp thực thi)**

* **LearningStrategy.java**

public interface LearningStrategy {

void learn(List<Word> words);

}

* **FlashcardStrategy.java**

class FlashcardStrategy implements LearningStrategy {

@Override

public void learn(List<Word> words) {

// ... logic học bằng flashcard

}

}

* **MultipleChoiceQuizStrategy.java**

class MultipleChoiceQuizStrategy implements LearningStrategy {

@Override

public void learn(List<Word> words) {

// ... logic học bằng trắc nghiệm

}

}

**Triển khai Mẫu Observer**

* **Trong VocabularyManager.java (Subject)**

public class VocabularyManager {

// ...

private List<ProgressObserver> observers;

public void registerObserver(ProgressObserver observer) {

observers.add(observer);

}

public void addWord(Word word) {

words.add(word);

// Notify all observers

for (ProgressObserver obs : observers) {

obs.onWordAdded(word);

}

}

// ...

}

* **Trong ProgressObserver.java (Interface và Concrete Observer)**

public interface ProgressObserver {

void onWordAdded(Word word);

}

class LearningProgress implements ProgressObserver {

private int total = 0;

@Override

public void onWordAdded(Word word) {

total++;

System.out.println("[Thong bao] Tu moi da duoc them! Tong: " + total);

}

}

**3.3. Kiểm thử qua Console**

**Chức năng: Thêm từ mới thành công**

* **Đầu vào:**
  + Chọn chức năng: 3
  + Nhập từ: sun
  + Nhập nghĩa: mat troi
  + Nhập ví dụ: The sun is shining.
* **Kết quả:**
* Nhap tu tieng Anh: sun
* Nhap nghia khong dau: mat troi
* Nhap vi du (neu co): The sun is shining.
* [Thong bao] Tu moi da duoc them! Tong: 9
* Tu da duoc them.

*(Observer đã hoạt động và thông báo tổng số từ đã tăng lên 9)*

**Chức năng: Thêm từ đã tồn tại**

* **Đầu vào:**
  + Chọn chức năng: 3
  + Nhập từ: apple
* **Kết quả:**
* Nhap tu tieng Anh: apple
* Tu da ton tai, khong them lai.

**Chức năng: Học theo Flashcard**

* **Đầu vào:**
  + Chọn chức năng: 1
  + Nhấn Enter
* **Kết quả:**
* === MENU ===
* 1. Hoc theo Flashcard
* ...
* Chon chuc nang: 1
* Tu: apple
* Nhan Enter de xem nghia...
* Nghia: qua tao
* ---
* Tu: book
* Nhan Enter de xem nghia...

**Chức năng: Học theo Quiz ABCD**

* **Đầu vào:**
  + Chọn chức năng: 2
  + Nhập đáp án: C (giả sử C là đáp án đúng)
* **Kết quả:**
* === MENU ===
* 1. Hoc theo Flashcard
* ...
* Chon chuc nang: 2
* Tu: computer
* A. con cho
* B. qua tao
* C. may tinh
* D. quyen sach
* Chon dap an (A/B/C/D): C
* Dung!
* ---

**4. Kết luận và Hướng phát triển Tương lai**

**4.1. Tóm tắt Thành tựu**

Dự án đã xây dựng thành công một ứng dụng học từ vựng cơ bản bằng Java, đáp ứng đầy đủ các yêu cầu chức năng và phi chức năng đã đề ra. Quan trọng hơn, dự án là một minh chứng rõ ràng cho việc áp dụng hiệu quả các nguyên lý thiết kế SOLID và các mẫu thiết kế phần mềm vào thực tế.

* **Singleton** đảm bảo tính nhất quán của dữ liệu.
* **Strategy** mang lại sự linh hoạt và khả năng mở rộng cho các phương pháp học.
* **Observer** tạo ra một cơ chế thông báo mềm dẻo, giảm thiểu sự phụ thuộc giữa các thành phần.

Việc áp dụng các kỹ thuật này đã giúp tạo ra một hệ thống có cấu trúc tốt, dễ bảo trì và sẵn sàng cho các nâng cấp trong tương lai.

**4.2. Tự đánh giá và Kinh nghiệm học được**

Qua quá trình thực hiện bài tập lớn, tôi đã có cơ hội chuyển hóa những kiến thức lý thuyết về SOLID và Design Patterns thành mã nguồn chạy được. Bài học lớn nhất là hiểu được "lý do" đằng sau mỗi quyết định thiết kế: không chỉ áp dụng mẫu một cách máy móc, mà phải nhận diện được vấn đề thiết kế cụ thể mà mẫu đó giải quyết. Ví dụ, việc so sánh cách triển khai "không dùng mẫu" và "có dùng mẫu" đã làm nổi bật rõ rệt lợi ích về tính linh hoạt và bảo trì của mẫu Strategy.

**4.3. Cải tiến và Hướng phát triển Tương lai**

Hệ thống hiện tại có thể được mở rộng và cải tiến theo nhiều hướng:

* **Lưu trữ dữ liệu (Persistence):** Hiện tại, kho từ vựng sẽ mất khi tắt ứng dụng. Có thể thêm chức năng lưu/tải danh sách từ vựng ra file (JSON, CSV) hoặc tích hợp với cơ sở dữ liệu (sử dụng mẫu **DAO - Data Access Object**).
* **Thêm nhiều Learning Strategies:** Xây dựng thêm các phương pháp học mới như TypingTestStrategy (người dùng gõ lại từ) hoặc FillInTheBlankStrategy (điền từ vào chỗ trống trong câu ví dụ).
* **Giao diện người dùng đồ họa (GUI):** Nâng cấp ứng dụng từ giao diện dòng lệnh lên giao diện đồ họa sử dụng JavaFX hoặc Swing (có thể áp dụng mẫu **MVC - Model View Controller**).
* **Theo dõi tiến trình học tập cá nhân:** Lưu lại kết quả các bài quiz để biết người dùng hay sai ở từ nào, từ đó đề xuất các từ cần ôn tập nhiều hơn (có thể dùng mẫu **State** để quản lý trạng thái học của từng từ).