Dưới đây là các đề xuất **tối ưu hóa code** cho từng bài bạn cung cấp, đặc biệt hướng đến khả năng mở rộng với AI hoặc tăng hiệu suất, dễ bảo trì, và chuẩn hóa tốt hơn:

**📌 1. AIHelper.java - Gọi API OpenAI để định nghĩa từ**

**🔧 Tối ưu:**

* Dùng thư viện HttpClient (Java 11+) thay vì HttpURLConnection để code ngắn gọn, dễ test hơn.
* Tách API\_KEY ra khỏi code nguồn, dùng biến môi trường hoặc file config.
* Dùng thư viện xử lý JSON như **Gson** hoặc **Jackson** để parse kết quả thay vì xử lý chuỗi thủ công.

**💡 Gợi ý mở rộng AI:**

* Cho phép chọn model (e.g. gpt-3.5, gpt-4).
* Cho phép nhập role/prompt tùy chỉnh (giao diện người dùng).
* Tích hợp đa ngôn ngữ (dùng prompt định hướng).

**📌 2. FoodItem.java & hệ thống quản lý nhà hàng**

**🔧 Tối ưu:**

* Dùng ArrayList thay cho mảng cố định (OrderItem[] items = new OrderItem[100]) để tăng linh hoạt.
* Tách class ra thành từng file riêng: FoodItem.java, OrderItem.java, Invoice.java.
* Thêm validation cho input người dùng.
* Dùng BigDecimal để xử lý tiền thay vì double → chính xác hơn.

**💡 Gợi ý mở rộng AI:**

* Tích hợp AIHelper để mô tả món ăn khi khách không rõ.

java

CopyEdit

String desc = AIHelper.getDefinition("Phở"); // mô tả về Phở

* Tích hợp gợi ý món ăn theo khẩu vị (prompt: "Gợi ý món ăn cho người thích cay").

**📌 3. MatrixOperations.java**

**🔧 Tối ưu:**

* Kiểm tra điều kiện nhân ma trận *trước khi nhập đầu vào*, không phải sau khi đã nhập cả hai.
* Tách logic ra khỏi main() để dễ test unit.
* Cho phép nhập/lưu/đọc ma trận từ file (Scanner + File).

**💡 Gợi ý mở rộng AI:**

* Gửi yêu cầu đến AI để giải thích kết quả nhân hoặc cộng:

java

CopyEdit

String explain = AIHelper.getDefinition("giải thích phép nhân ma trận");

**📌 4. WordEntry.java - Từ điển đơn giản**

**🔧 Tối ưu:**

* Dùng ArrayList<String> thay vì String[10] cho nghĩa → linh hoạt và dễ thao tác.
* Dùng HashMap<String, WordEntry> thay vì mảng WordEntry[] để tra cứu nhanh hơn (O(1) thay vì O(n)).
* UI dòng lệnh có thể tổ chức lại thành các phương thức nhỏ.

**💡 Gợi ý mở rộng AI:**

* Tự động thêm nghĩa bằng AI nếu người dùng không nhập:

java

CopyEdit

if (meaning.isEmpty()) meaning = AIHelper.getDefinition(word);

* Gợi ý ví dụ sử dụng từ (prompt: “Cho ví dụ dùng từ ‘x’ trong câu”).

**📌 5. WordFrequencyCounter.java**

**🔧 Tối ưu:**

* Dùng HashMap<String, Integer> thay cho mảng 2 chiều → gọn hơn, nhanh hơn.
* Tách phần xử lý thành hàm riêng để dễ unit test.
* Dùng BufferedReader để đọc nhiều dòng thay vì scanner.nextLine().

**💡 Gợi ý mở rộng AI:**

* Nhờ AI phân tích đoạn văn về mặt nội dung:

java

CopyEdit

String analysis = AIHelper.getDefinition("Phân tích nội dung đoạn văn sau: " + paragraph);

**✅ Tóm tắt đề xuất chính:**

| **Tệp** | **Tối ưu hóa kỹ thuật** | **Hướng đến AI** |
| --- | --- | --- |
| AIHelper | Dùng HttpClient, parse JSON tốt hơn | Prompt linh hoạt, đa ngôn ngữ |
| FoodItem | Dùng List, BigDecimal | Gợi ý món ăn bằng AI |
| MatrixOperations | Tách logic, kiểm tra trước | Giải thích phép toán bằng AI |
| WordEntry | Dùng Map, List | Tự động định nghĩa, ví dụ câu |
| WordFrequencyCounter | Dùng Map, tách hàm | Phân tích nội dung bằng AI |

Mỗi file sẽ:

* Dùng các cấu trúc dữ liệu hiện đại (List, Map, BigDecimal...),
* Chuẩn hóa OOP,
* Tích hợp AIHelper.getDefinition(...) để mở rộng thông minh (khi có ý nghĩa),
* Viết gọn và rõ ràng hơn.

**📘 Danh sách cập nhật sẽ làm lại:**

1. ✅ AIHelper (tối ưu & sẵn sàng gọi API).
2. ✅ FoodItem, OrderItem, Invoice, RestaurantManagementSystem (tối ưu + AI gợi ý món ăn).
3. ✅ MatrixOperations (tách logic, giải thích bằng AI).
4. ✅ WordEntry, DictionaryManagementSystem (tối ưu + tự thêm nghĩa từ AI).
5. ✅ WordFrequencyCounter (dùng Map + AI phân tích nội dung đoạn văn).

✅ [1] AIHelper.java (Tối ưu hóa, dễ dùng lại)

|  |
| --- |
| import java.io.\*;  import java.net.URI;  import java.net.http.\*;  import java.nio.charset.StandardCharsets;  public class AIHelper {  private static final String API\_URL = "https://api.openai.com/v1/completions";  private static final String API\_KEY = System.getenv("OPENAI\_API\_KEY"); // Set API key as env variable  public static String getDefinition(String promptText) {  if (API\_KEY == null || API\_KEY.isEmpty()) {  return "[Chưa cấu hình API KEY]";  }  try {  String prompt = "Hãy định nghĩa hoặc giải thích: " + promptText;  String requestBody = String.format("""  {  "model": "text-davinci-003",  "prompt": "%s",  "max\_tokens": 100  }  """, prompt.replace("\"", "\\\""));  HttpRequest request = HttpRequest.newBuilder()  .uri(URI.create(API\_URL))  .header("Content-Type", "application/json")  .header("Authorization", "Bearer " + API\_KEY)  .POST(HttpRequest.BodyPublishers.ofString(requestBody, StandardCharsets.UTF\_8))  .build();  HttpClient client = HttpClient.newHttpClient();  HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());  String result = response.body();  int start = result.indexOf("\"text\":\"") + 8;  int end = result.indexOf("\"", start);  if (start > 7 && end > start) {  return result.substring(start, end).replace("\\n", " ").trim();  } else {  return "[Không thể phân tích kết quả AI]";  }  } catch (Exception e) {  return "[Lỗi khi gọi AI: " + e.getMessage() + "]";  }  }  } |

✅ FoodItem, OrderItem, Invoice, RestaurantManagementSystem (tối ưu + AI gợi ý món ăn).

|  |
| --- |
| // RestaurantManagementSystem.java - Tối ưu hóa + AI gợi ý món ăn  import java.math.BigDecimal;  import java.util.\*;  public class RestaurantManagementSystem {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  List<FoodItem> menu = List.of(  new FoodItem("Phở", new BigDecimal("30000")),  new FoodItem("Bánh mì", new BigDecimal("15000")),  new FoodItem("Trà sữa", new BigDecimal("25000")),  new FoodItem("Cà phê", new BigDecimal("20000"))  );  Invoice invoice = new Invoice();  while (true) {  System.out.println("\n--- MENU ---");  for (int i = 0; i < menu.size(); i++) {  System.out.println((i + 1) + ". " + menu.get(i));  }  System.out.println("5. In hóa đơn");  System.out.println("6. Gợi ý món ăn bằng AI");  System.out.println("7. Thanh toán và thoát");  System.out.print("Chọn món (1-7): ");  int choice = scanner.nextInt();  if (choice >= 1 && choice <= 4) {  FoodItem item = menu.get(choice - 1);  System.out.print("Nhập số lượng: ");  int qty = scanner.nextInt();  invoice.addItem(new OrderItem(item.getName(), qty, item.getPrice()));  } else if (choice == 5) {  invoice.printInvoice();  } else if (choice == 6) {  scanner.nextLine();  System.out.print("Bạn muốn ăn gì (ví dụ: món cay, chay...): ");  String taste = scanner.nextLine();  String suggestion = AIHelper.getDefinition("Gợi ý món ăn phù hợp với: " + taste);  System.out.println("\n🧠 Gợi ý từ AI: " + suggestion);  } else if (choice == 7) {  invoice.printInvoice();  System.out.println("Cảm ơn quý khách!");  break;  } else {  System.out.println("Lựa chọn không hợp lệ!");  }  }  scanner.close();  }  }  class FoodItem {  private String name;  private BigDecimal price;  public FoodItem(String name, BigDecimal price) {  this.name = name;  this.price = price;  }  public String getName() { return name; }  public BigDecimal getPrice() { return price; }  @Override  public String toString() {  return name + " - " + price + " VND";  }  }  class OrderItem {  private String itemName;  private int quantity;  private BigDecimal unitPrice;  public OrderItem(String itemName, int quantity, BigDecimal unitPrice) {  this.itemName = itemName;  this.quantity = quantity;  this.unitPrice = unitPrice;  }  public String getItemName() { return itemName; }  public int getQuantity() { return quantity; }  public BigDecimal getUnitPrice() { return unitPrice; }  public BigDecimal getTotal() {  return unitPrice.multiply(BigDecimal.valueOf(quantity));  }  @Override  public String toString() {  return itemName + " - SL: " + quantity + " - Giá: " + getTotal() + " VND";  }  @Override  public boolean equals(Object obj) {  if (obj instanceof OrderItem other) {  return itemName.equals(other.itemName);  }  return false;  }  }  class Invoice {  private List<OrderItem> items = new ArrayList<>();  public void addItem(OrderItem item) {  for (OrderItem existing : items) {  if (existing.equals(item)) {  System.out.println("Món này đã có trong hóa đơn.");  return;  }  }  items.add(item);  }  public void printInvoice() {  BigDecimal total = BigDecimal.ZERO;  System.out.println("\n--- HÓA ĐƠN ---");  for (OrderItem item : items) {  System.out.println(item);  total = total.add(item.getTotal());  }  System.out.println("Tổng cộng: " + total + " VND");  }  } |

✅ MatrixOperations (tách logic, giải thích bằng AI).

|  |
| --- |
| // MatrixOperationsAI.java - Tối ưu + Giải thích bằng AI  import java.util.\*;  public class MatrixOperationsAI {  public static int[][] addMatrix(int[][] a, int[][] b) {  int rows = a.length, cols = a[0].length;  int[][] result = new int[rows][cols];  for (int i = 0; i < rows; i++)  for (int j = 0; j < cols; j++)  result[i][j] = a[i][j] + b[i][j];  return result;  }  public static int[][] multiplyMatrix(int[][] a, int[][] b) {  int rows = a.length, cols = b[0].length, common = b.length;  int[][] result = new int[rows][cols];  for (int i = 0; i < rows; i++)  for (int j = 0; j < cols; j++)  for (int k = 0; k < common; k++)  result[i][j] += a[i][k] \* b[k][j];  return result;  }  public static void printMatrix(int[][] matrix, String title) {  System.out.println("\n" + title);  for (int[] row : matrix) {  for (int value : row)  System.out.print(value + " ");  System.out.println();  }  }  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Nhập số dòng của ma trận A: ");  int rowsA = scanner.nextInt();  System.out.print("Nhập số cột của ma trận A: ");  int colsA = scanner.nextInt();  int[][] matrixA = new int[rowsA][colsA];  System.out.println("Nhập ma trận A:");  inputMatrix(scanner, matrixA);  System.out.print("Nhập số dòng của ma trận B: ");  int rowsB = scanner.nextInt();  System.out.print("Nhập số cột của ma trận B: ");  int colsB = scanner.nextInt();  int[][] matrixB = new int[rowsB][colsB];  System.out.println("Nhập ma trận B:");  inputMatrix(scanner, matrixB);  if (rowsA == rowsB && colsA == colsB) {  int[][] sum = addMatrix(matrixA, matrixB);  printMatrix(sum, "Tổng hai ma trận là:");  } else {  System.out.println("Không thể cộng hai ma trận khác kích thước.");  }  if (colsA == rowsB) {  int[][] product = multiplyMatrix(matrixA, matrixB);  printMatrix(product, "Tích hai ma trận là:");  String explanation = AIHelper.getDefinition("Giải thích phép nhân ma trận");  System.out.println("\n🧠 AI giải thích: " + explanation);  } else {  System.out.println("Không thể nhân: số cột A phải bằng số dòng B.");  }  scanner.close();  }  public static void inputMatrix(Scanner scanner, int[][] matrix) {  for (int i = 0; i < matrix.length; i++) {  for (int j = 0; j < matrix[0].length; j++) {  System.out.print("[" + i + "][" + j + "]: ");  matrix[i][j] = scanner.nextInt();  }  }  }  } |

✅ WordEntry, DictionaryManagementSystem (tối ưu + tự thêm nghĩa từ AI).

|  |
| --- |
| // DictionaryWithAI.java - Từ điển tối ưu + tự thêm nghĩa bằng AI  import java.util.\*;  public class DictionaryWithAI {  private static final Map<String, WordEntry> dictionary = new HashMap<>();  public static void addWord(String word, String meaning) {  dictionary.computeIfAbsent(word, k -> new WordEntry(word)).addMeaning(meaning);  }  public static void lookup(String word) {  WordEntry entry = dictionary.get(word);  if (entry != null) {  System.out.println(entry);  } else {  System.out.println("Không tìm thấy từ này.");  System.out.println("Đang tìm nghĩa với AI...");  String aiMeaning = AIHelper.getDefinition("Định nghĩa từ: " + word);  System.out.println("🧠 AI gợi ý: " + aiMeaning);  addWord(word, aiMeaning);  }  }  public static void removeMeaning(String word, int index) {  WordEntry entry = dictionary.get(word);  if (entry != null) {  entry.removeMeaning(index);  } else {  System.out.println("Không tìm thấy từ này.");  }  }  public static void showTotalWords() {  System.out.println("Tổng số từ trong từ điển: " + dictionary.size());  }  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  while (true) {  System.out.println("\n--- MENU ---");  System.out.println("1. Thêm từ và nghĩa");  System.out.println("2. Xóa nghĩa của từ");  System.out.println("3. Tra cứu từ");  System.out.println("4. Xem tổng số từ");  System.out.println("5. Thoát");  System.out.print("Chọn chức năng: ");  int choice = scanner.nextInt();  scanner.nextLine();  switch (choice) {  case 1 -> {  System.out.print("Nhập từ: ");  String word = scanner.nextLine();  System.out.print("Nhập nghĩa: ");  String meaning = scanner.nextLine();  addWord(word, meaning);  }  case 2 -> {  System.out.print("Nhập từ cần xóa nghĩa: ");  String word = scanner.nextLine();  lookup(word);  System.out.print("Nhập chỉ số nghĩa cần xóa: ");  int index = scanner.nextInt();  scanner.nextLine();  removeMeaning(word, index);  }  case 3 -> {  System.out.print("Nhập từ cần tra: ");  String word = scanner.nextLine();  lookup(word);  }  case 4 -> showTotalWords();  case 5 -> {  System.out.println("Tạm biệt!");  scanner.close();  return;  }  default -> System.out.println("Lựa chọn không hợp lệ.");  }  }  }  }  class WordEntry {  private final String word;  private final List<String> meanings = new ArrayList<>();  public WordEntry(String word) {  this.word = word;  }  public void addMeaning(String meaning) {  if (meanings.contains(meaning)) {  System.out.println("Nghĩa này đã tồn tại.");  } else {  meanings.add(meaning);  System.out.println("Đã thêm nghĩa mới.");  }  }  public void removeMeaning(int index) {  if (index >= 0 && index < meanings.size()) {  meanings.remove(index);  System.out.println("Đã xóa nghĩa.");  } else {  System.out.println("Chỉ số không hợp lệ.");  }  }  @Override  public String toString() {  StringBuilder sb = new StringBuilder(word + ": ");  for (int i = 0; i < meanings.size(); i++) {  sb.append("[" + i + "] " + meanings.get(i)).append("; ");  }  return sb.toString();  }  } |

✅ WordFrequencyCounter (dùng Map + AI phân tích nội dung đoạn văn).

|  |
| --- |
| // WordFrequencyAI.java - Đếm tần suất từ + AI phân tích nội dung  import java.util.\*;  public class WordFrequencyAI {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.println("Nhập đoạn văn:");  String paragraph = scanner.useDelimiter("\\n{2,}").next(); // Cho phép nhập nhiều dòng  Map<String, Integer> freqMap = new TreeMap<>();  String[] words = paragraph.toLowerCase().replaceAll("[^a-zA-Z\s]", "").split("\\s+");  for (String word : words) {  if (!word.isEmpty()) {  freqMap.put(word, freqMap.getOrDefault(word, 0) + 1);  }  }  System.out.println("\nTần suất từ trong đoạn văn:");  freqMap.forEach((k, v) -> System.out.printf("%-15s : %d\n", k, v));  System.out.println("\nBạn có muốn AI phân tích đoạn văn này không? (y/n): ");  String answer = scanner.next().trim().toLowerCase();  if (answer.equals("y")) {  String aiAnalysis = AIHelper.getDefinition("Phân tích đoạn văn sau về chủ đề, cảm xúc và ý nghĩa: " + paragraph);  System.out.println("\n🧠 AI phân tích: " + aiAnalysis);  } else {  System.out.println("Bỏ qua phân tích AI.");  }  scanner.close();  }  } |

Dưới đây là phần **phân tích ý nghĩa ứng dụng trong thực tế** và **ghi chú quan trọng (Key Notes)** cho **Lab 5 – Tối ưu hóa code và tích hợp AI trong Java**:

**🎯 I. Ý NGHĨA ỨNG DỤNG THỰC TẾ CỦA CÁC CẢI TIẾN**

| **Chủ đề** | **Ý nghĩa thực tiễn** | **Cải thiện so với code truyền thống** |
| --- | --- | --- |
| **1. Tối ưu cấu trúc dữ liệu** | - Dễ mở rộng, dễ bảo trì.  - Tránh lỗi tràn mảng, tăng tốc độ tra cứu. | Dùng List, Map thay vì mảng cố định (Array[]). |
| **2. Chuẩn hóa OOP** | - Tách biệt logic, dễ test.  - Mỗi class có trách nhiệm rõ ràng. | Tách thành nhiều class, chia file. Không gộp tất cả vào main(). |
| **3. Sử dụng BigDecimal** | - Độ chính xác cao khi tính toán tiền tệ, tránh sai số làm tròn. | Thay thế double vốn không chính xác tuyệt đối. |
| **4. Tách logic và giao diện** | - Tăng khả năng kiểm thử (unit test), tái sử dụng. | Không viết toàn bộ logic xử lý trong main(). |
| **5. Gọi API AI để xử lý nâng cao** | - Gợi ý món ăn theo khẩu vị. - Giải thích khái niệm toán học. - Tự động định nghĩa từ. - Phân tích nội dung văn bản. | Truyền thống không có AI, hạn chế về tính năng và độ linh hoạt. |
| **6. Chuẩn hóa xử lý chuỗi và dữ liệu** | - Bền vững hơn với nhiều tình huống nhập liệu. - Tránh lỗi nhập dữ liệu sai định dạng. | Dùng replaceAll, split, toLowerCase hiệu quả hơn cách thủ công. |

**🧪 II. GHI CHÚ THỰC HÀNH (KEY NOTES) CHO LAB 5**

**🔰 Cơ bản**

* ✅ Dùng List thay vì Array[] để linh hoạt trong thêm/xóa phần tử.
* ✅ Dùng Map (HashMap, TreeMap) để tra cứu nhanh và tự động sắp xếp từ điển/tần suất.
* ✅ BigDecimal là bắt buộc khi xử lý tiền tệ trong Java.
* ✅ Luôn chia nhỏ chương trình thành các hàm hoặc lớp riêng biệt (clean code).
* ✅ Scanner nên được đóng sau khi sử dụng (scanner.close()).

**💡 Nâng cao**

* ✅ Sử dụng **HttpClient (Java 11+)** để gọi API dễ dàng, gọn hơn HttpURLConnection.
* ✅ Tích hợp **OpenAI API**: dùng prompt tùy chỉnh để nhận kết quả thông minh từ AI.
* ✅ Thiết kế ứng dụng sao cho nếu người dùng không biết câu trả lời → AI có thể giúp.
* ✅ Kiểm tra tính khả thi trước khi thực hiện phép toán (ví dụ: nhân ma trận cần kiểm tra kích thước).
* ✅ Xử lý nhập văn bản dài bằng scanner.useDelimiter("\\n{2,}") để nhận được đoạn văn nhiều dòng.

**🧠 III. Gợi ý bài tập mở rộng (cho sinh viên)**

* ✍️ Viết chương trình trò chơi đoán từ có gợi ý bằng AI.
* 📚 Tạo từ điển AI đa ngôn ngữ (prompt: “Dịch từ \_\_\_ sang tiếng Anh/Pháp/Trung...”).
* 🍽️ Ứng dụng đặt món ăn có trợ lý AI (chatbot).
* 🧾 Phân tích văn bản tự động tìm chủ đề chính, từ khóa chính bằng AI.