LAB 1: BÀI TẬP CODE READING

Hướng dẫn thực hiện

Bước 1: Xác định mục tiêu của bài tập

- Hiểu rõ chức năng của từng đoạn mã.
- Xác định lỗi tiềm ẩn (nếu có).
- Đề xuất cải tiến về logic, hiệu suất, hoặc khả năng đọc hiểu.
- Thực hành kỹ năng phân tích và cải tiến mã nguồn.

Bước 2: Quy trình thực hiện

1. Đọc và hiểu mã nguồn

- Sinh viên đọc đoạn mã một cách tổng thể để hiểu mục đích chính của từng hàm.
- Phân tích vai trò của từng phần:

2. Tìm kiếm lỗi hoặc vấn đề tiềm ẩn

3. Đề xuất cải tiến

Các bài tập này giúp sinh viên phát triển kỹ năng phân tích, đánh giá mã nguồn, và học cách viết mã chất lượng cao.

Bài tập 1.

Đoạn mã:

```
import java.util.List;

public class ScoreCalculator {
    public static double calculateAverage(List<Integer> scores) {
        int total = 0;
        for (int score : scores) {
            total += score;
        }
        return (double) total / scores.size(); // Tính điểm trung bình
    }
    public static int findHighestScore(List<Integer> scores) {
```

```
        int highest = scores.get(0);
        for (int score : scores) {
            if (score > highest) {
                highest = score;
            }
        }
        return highest;
    }

    public static void main(String[] args) {
        List<Integer> scores = List.of(85, 90, 78, 92, 88);
        double averageScore = calculateAverage(scores);
        int highestScore = findHighestScore(scores);
        System.out.println("Average score: " + averageScore);
        System.out.println("Highest score: " + highestScore);
    }
}
```

Yêu cầu:

1. Đọc và hiểu đoạn mã trên.
2. Xác định có lỗi nào trong mã (gợi ý: điều gì sẽ xảy ra nếu danh sách scores trống?).
3. Đề xuất cải tiến hoặc tối ưu hóa đoạn mã (ví dụ: sử dụng hàm tích hợp max thay cho vòng lặp tìm điểm cao nhất).

Mục tiêu:

- Hiểu logic và cấu trúc của đoạn mã.
- Phát hiện lỗi, nếu có, và đề xuất cải tiến.

<u>Bài tập 2: Phát hiện lỗi logic trong mã</u>

Đoạn mã:

```
public class FactorialCalculator {
   public static int calculateFactorial(int n) {
      if (n < 0) {
         throw new IllegalArgumentException("Number must be non-negative.");
      }
      int result = 1;
      for (int i = 1; i <= n; i++) {
         result *= i;
      }
      return result;
   }

   public static void main(String[] args) {
      int number = 5;
      System.out.println("Factorial of " + number + " is: " + calculateFactorial(number));
   }
}
```

Yêu cầu:

1. Đọc đoạn mã và xác định xem nó hoạt động đúng không.
2. Kiểm tra cách xử lý giá trị n = 0. Chương trình có hoạt động chính xác không?
3. Đề xuất cải tiến để mã trở nên an toàn hơn và hiệu quả hơn.

Mục tiêu:

- Sinh viên học cách kiểm tra điều kiện biên và đảm bảo độ chính xác của chương trình.

Bài tập 3: Tối ưu hóa và cải thiện hiệu suất

Đoạn mã:

```
public class ArraySearch {
   public static int findElement(int[] arr, int target) {
      for (int i = 0; i < arr.length; i++) {
         if (arr[i] == target) {
            return i; // Trả về chỉ số đầu tiên tìm thấy
         }
      }
      return -1; // Không tìm thấy
```

```
    }

    public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40, 50};
        int target = 30;
        System.out.println("Index of target: " + findElement(numbers, target));
    }
}
```

Yêu cầu:

1. Đọc mã và kiểm tra tính chính xác.
2. Đề xuất cách tối ưu hóa nếu mảng đã được sắp xếp (gợi ý: sử dụng thuật toán tìm kiếm nhị phân).
3. Thêm kiểm tra để đảm bảo mảng không null và không rỗng.

Mục tiêu:

- Sinh viên học cách xác định cơ hội cải tiến hiệu suất và đảm bảo an toàn mã.

Bài tập 4: Xử lý các ngoại lệ và cải thiện khả năng mở rộng

Đoạn mã mẫu:

java
CopyEdit

```
import java.util.Scanner;

public class DivisionCalculator {
    public static double divide(int a, int b) {
        return a / b; // Phép chia cơ bản
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the numerator: ");
        int numerator = scanner.nextInt();
        System.out.print("Enter the denominator: ");
        int denominator = scanner.nextInt();

        System.out.println("Result: " + divide(numerator, denominator));
    }
}
```

Yêu cầu:

1. Đọc mã và xác định các vấn đề có thể xảy ra (ví dụ: chia cho 0, nhập không hợp lệ).
2. Thêm cơ chế xử lý ngoại lệ (try-catch) để xử lý lỗi một cách an toàn.
3. Đề xuất cải tiến để chương trình có thể chia nhiều cặp số mà không cần khởi động lại.

Mục tiêu:

- Sinh viên học cách xử lý ngoại lệ và cải thiện tính ổn định của chương trình.