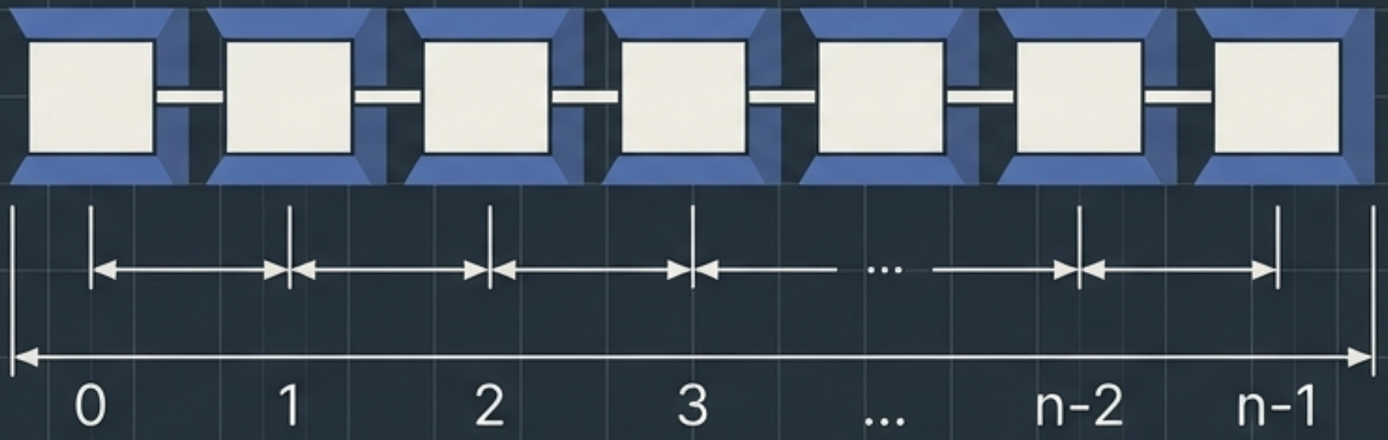
Xử lý mảng trong Java: Bản thiết kế từ ý tưởng đến hoàn chỉnh

Xây dựng tư duy thuật toán, tách module và kiểm soát dữ liệu an toàn



Phạm Xuân Hoài | Error404-Labs

5 Yêu Cầu Của Bài Toán Nền Tảng

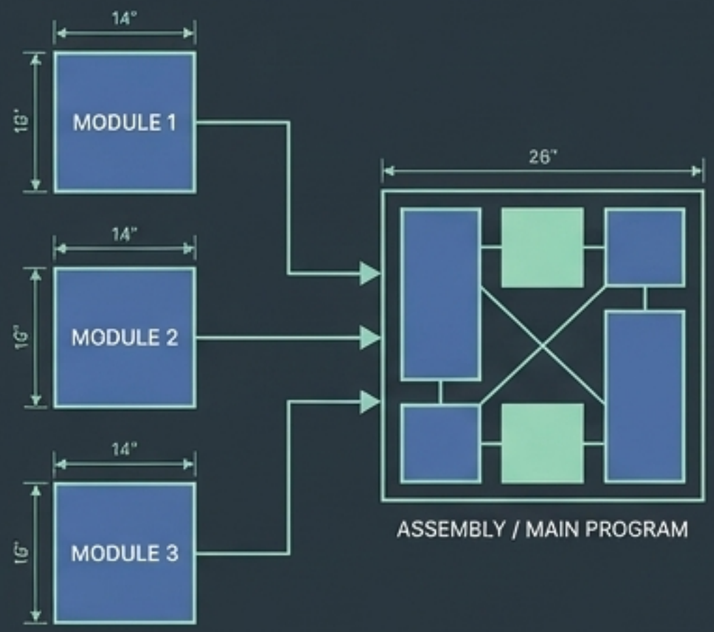
1.  Nhập số nguyên n (số lượng phần tử) - Đảm bảo an toàn, không crash.
2.  Nạp n số nguyên vào mảng dữ liệu.
3.  In mảng ra màn hình để kiểm tra.
4.  Sắp xếp mảng theo thứ tự tăng dần.
5. 

Quét và trích xuất giá trị Lớn nhất (Max) & Nhỏ nhất (Min).

Bài toán này là chìa khóa để làm chủ 3 kỹ năng cốt lõi: Mảng (Array), Vòng lặp (Loops), và Tư duy tách hàm (Modularization).

Triết lý thiết kế: Đừng viết code, hãy chế tạo Cỗ máy Module

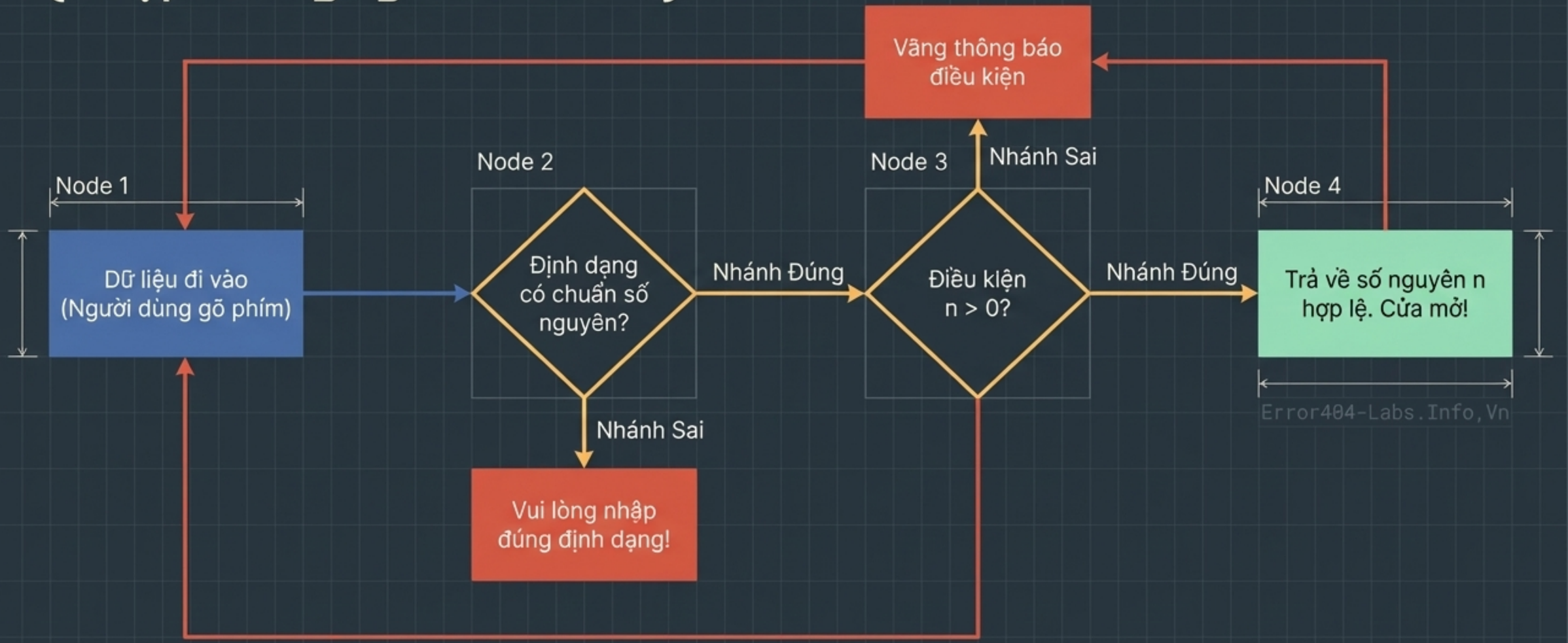
Cách code nghiệp dư: Viết mọi thứ vào hàm main().
Khó đọc, khó sửa, lặp lại code liên tục.



- Tư duy kỹ sư: Tách chương trình thành các Module độc lập.
- Mỗi hàm là một cỗ máy chỉ làm một nhiệm vụ duy nhất.
- Có Đầu vào (Input) và Đầu ra (Output) rõ ràng.
- Có thể tái sử dụng ở bất cứ đâu.

Toàn bộ chương trình của chúng ta sẽ được lắp ráp từ 6 cỗ máy con.

Module 1: Cổng an ninh kiểm duyệt (Nhập số nguyên an toàn)



Mã nguồn Module 1: Hàm nhapSoNguyen()

```java
static int nhapSoNguyen(String label, String condition, int mode) {
    while (true) {
        try {
            System.out.print(label);
            int n = Integer.parseInt(sc.nextLine());
            if (n < 1 && mode == 0) {
                System.out.println("\n" + condition);
                continue;
            }
            return n;
        } catch (Exception e) {
            System.out.println("\n Vui lòng nhập đúng định dạng!");
        }
    }
}
```

Error404-Labs.Info,Vn

Vòng lặp vô tận, chỉ thoát ra (return) khi dữ liệu đã chuẩn chỉnh.

Tấm khiên chặn lỗi crash chương trình khi người dùng cố tình nhập chữ cái thay vì số.

Tham số linh hoạt giúp hàm này có thể dùng chung cho cả việc nhập n (cần > 0) và nhập phần tử mảng (không cần điều kiện).

Module 2 & 3: Dây chuyền Nạp & Xuất Dữ Liệu

Duyệt qua từng vị trí từ 0 đến n-1.

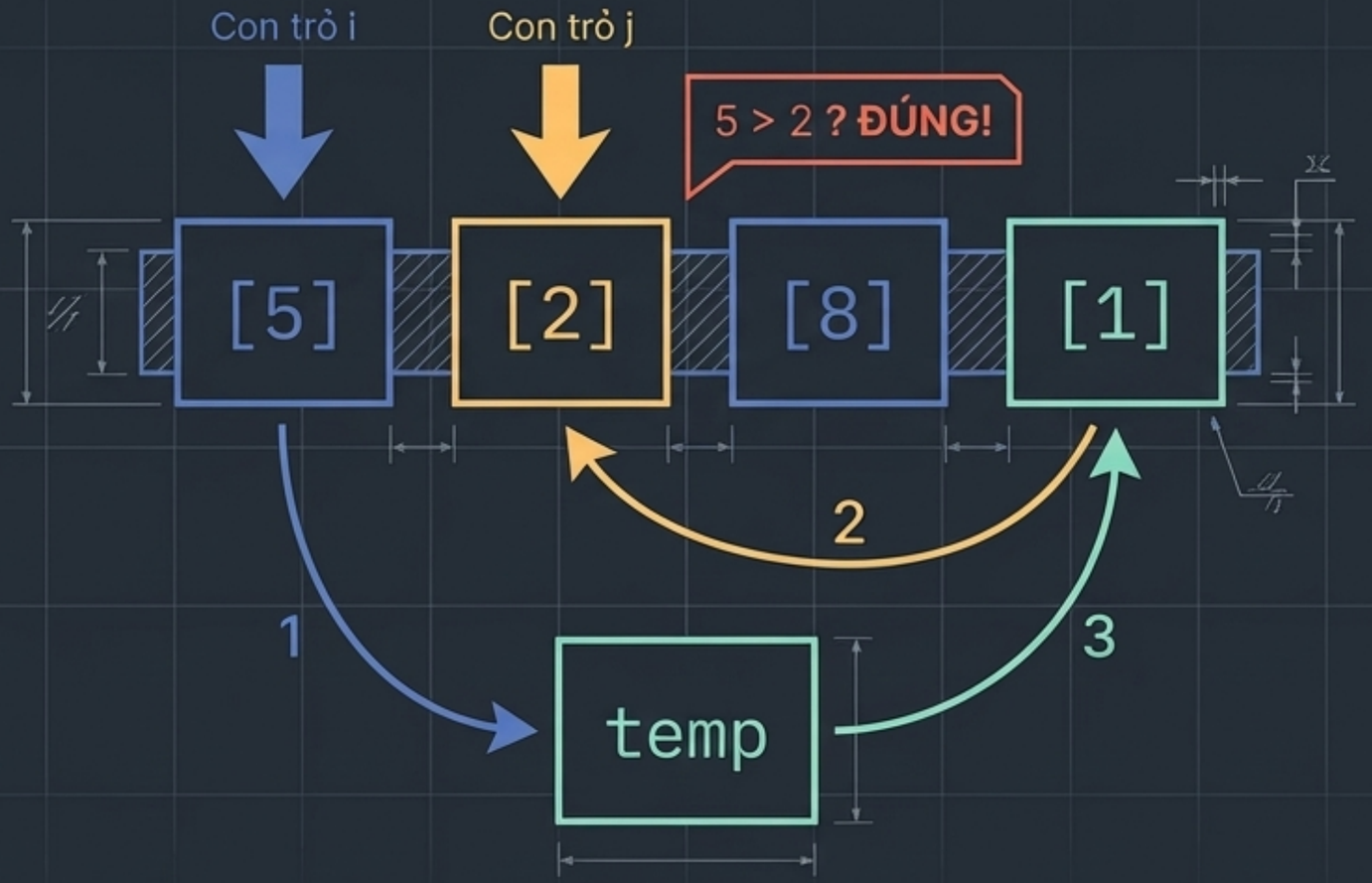
Phép màu của tái sử dụng! Ở đây ta gọi lại chính hàm **nhapSoNguyen()** vừa tạo ở Module 1. Code cực kỳ ngắn gọn và an toàn.

```
static void nhapMang(int a[]) {
  for (int i = 0; i < a.length; i++) {
    a[i] = nhapSoNguyen("a[" + i + "] = ", "", 1);
  }
}
```


In tiêu đề, duyệt mảng và in từng phần tử kèm khoảng trắng.

```
static void xuatMang(int a[], String label) {
   System.out.println("\n" + label);
   for (int i = 0; i < a.length; i++) {
      System.out.print(a[i] + " ");
   }
}
```

Module 4: Cơ chế Hoán Đổi của thuật toán Sắp Xếp



Thuật toán sử dụng 2 vòng lặp lồng nhau. i đứng yên, j chạy dò tìm các phần tử phía sau. Nếu phần tử trước lớn hơn phần tử sau -> Đảo vị trí thông qua biến trung gian temp.

Mã nguồn Module 4: Hàm sapXepTang()

Độ phức tạp: $O(n^2)$ - Đơn giản, trực quan, hoàn hảo cho người mới bắt đầu.


```
static void sapXepTang(int a[]) {
    for (int i = 0; i < a.length - 1; i++) {
        for (int j = i + 1; j < a.length; j++) {
            if (a[i] > a[j]) {
                int temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
}
```

Con trỏ i chốt vị trí hiện tại.

Điều kiện quyết định việc đổi chỗ.

Bộ ba dòng lệnh kinh điển để hoán vị 2 giá trị.

Con trỏ j quét toàn bộ các phần tử đứng sau i.

Module 5 & 6: Bộ đôi Cỗ máy Quét Cực Đại & Cực Tiểu

Giả sử phần tử đầu tiên (a[0]) là kết quả. Duyệt mảng từ vị trí 1 và cập nhật nếu tìm thấy giá trị thỏa mãn.

timMax(int a[])

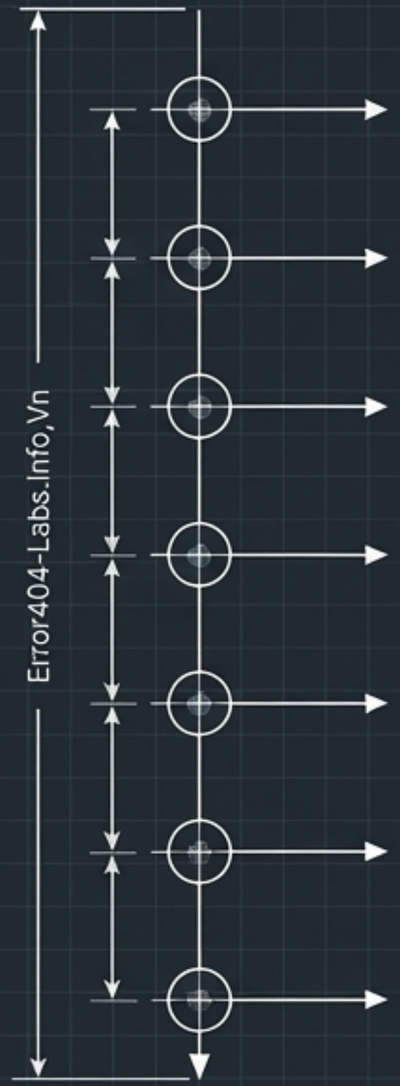
```
static int timMax(int a[]) {
    int max = a[0];
    for (int i = 1; i < a.length; i++) {
        if (a[i] > max) {
            max = a[i];
        }
    }
    return max;
}
```

timMin(int a[])

```
static int timMin(int a[]) {
    int min = a[0];
    for (int i = 1; i < a.length; i++) {
        if (a[i] < min) {
            min = a[i];
        }
    }
    return min;
}
```

Cấu trúc thuật toán hoàn toàn giống nhau 100%, chỉ đảo ngược duy nhất dấu > thành < ở biểu thức so sánh.

Lắp ráp Dây Chuyền: Hàm main() điều khiển trung tâm



Bước 1: Gọi nhapSoNguyen -> Khởi tạo biến n.

Bước 2: new int[n] -> Đúc ra một mảng trống có n chỗ chứa.

Bước 3: Gọi nhapMang -> Nạp dữ liệu vào mảng trống.

Bước 4: Gọi xuatMang -> Hiển thị mảng nguyên bản.

Bước 5: Gọi sapXepTang -> Biến đổi mảng.

Bước 6: Gọi xuatMang -> Hiển thị mảng sau khi sắp xếp.

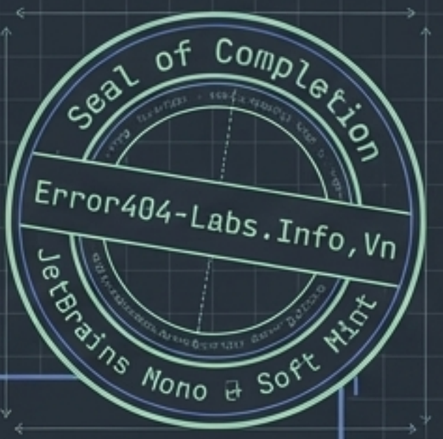
Bước 7: Gọi timMax, timMin -> In ra 2 giá trị cực hạn.

```java
public static void main(String[] args) {
    int n = nhapSoNguyen("Nhập số nguyên: ",
        "Vui lòng nhập n > 0!", 0);
    int a[] = new int[n];
    nhapMang(a);
    xuatMang(a, "Mảng đã nhập:");
    sapXepTang(a);
    xuatMang(a, "Mảng đã sắp xếp tăng:");
    System.out.println("\n Max = " + timMax(a));
    System.out.println("Min = " + timMin(a));
}
```


JetBrains Mono

Hàm main() giờ đây không chứa các vòng lặp lằng nhằng. Nó chỉ làm nhiệm vụ của một người Nhạc Trưởng: Gọi tên từng cỗ máy theo đúng thứ tự.

Hoàn tất Bản Thiết Kế: Nền tảng cho tương lai



Trang bị cốt lõi

- [x] Làm chủ các thao tác vòng lặp và Mảng (`Array`).
- [x] Biết cách thiết lập Cửa chốt an toàn cho Input với `try-catch`.
- [x] Hiểu sâu sắc tư duy Chia để trị (Tách `module` / `Tách hàm`).
- [x] Nắm được logic của thuật toán Sắp xếp, Tìm Max/Min.

Trạm tiếp theo

Bản thiết kế này là bước đệm bắt buộc trước khi bạn tiến tới:

- Cấu trúc dữ liệu nâng cao (`Danh sách liên kết`, `Cây`, `Đồ thị`).
- Thuật toán Sắp xếp tối ưu hơn (`Quick Sort`, `Merge Sort`).
- Thuật toán Tìm kiếm nhị phân (`Binary Search`).

Tiếp tục rèn luyện kỹ năng tại `Error404-Labs.Info,Vn`.
Lập trình không phải là học thuộc lòng code, mà là làm chủ tư duy lắp ráp!