Bài tập 1: Trong một hệ thống quản lý dữ liệu cảm biến, yêu cầu đặt ra là chỉ được phép tồn tại một thể hiện duy nhất của đối tượng **SensorManager** trong toàn bộ hệ thống. Đối tượng **SensorManager có nhiệm vụ chính là**:

1. Quản lý và thu thập dữ liệu từ các cảm biến (**SensorNode**).
2. Đưa dữ liệu lên cơ sở dữ liệu một cách hiệu quả và nhất quán.

Nếu không đảm bảo được răng chỉ có một đối tượng SensorManager duy nhất được tạo ra, hệ thống có thể gặp các vấn đề nghiêm trọng như sau:

1. **Dữ liệu bị trùng lặp**: Nhiều đối tượng SensorManager có thể thu thập cùng một dữ liệu từ cảm biến, dẫn đến việc lưu trữ dữ liệu lặp lại trong cơ sở dữ liệu.
2. **Xung đột tài nguyên**: Các đối tượng SensorManager khác nhau có thể cùng truy cập và thao tác trên các tài nguyên chung, gây ra xung đột hoặc lỗi không mong muốn.
3. **Khó kiểm soát**: Việc quản lý và theo dõi hành vi của nhiều đối tượng SensorManager trở nên

phức tạp, làm giảm tính ổn định và khả năng bảo trì của hệ thống

Yêu cầu:

1. Đề xuất và triển khai giải pháp để đảm bảo rằng chỉ có một thể hiện duy nhất của đối tượng SensorManager được tạo ra trong toàn bộ hệ thống.
2. Giải thích cách giải pháp của bạn giúp giải quyết các vấn đề đã nêu.

Gợi ý: Sử dụng **Singleton Pattern**

Bài tập 2: Bạn đang phát triển một hệ thống nhúng để giao tiếp với các cảm biến khác nhau. Ban đầu, hệ thống chỉ được thiết kế để giao tiếp và lấy dữ liệu từ cảm biến nhiệt độ. Tuy nhiên, theo thời gian, khách hàng có thêm yêu cầu mới: hệ thống cần có khả năng đo lường các thông số khác, như độ ẩm không khí, áp suất, hoặc ánh sáng.

Việc tích hợp cảm biến mới (ví dụ: cảm biến độ ẩm) đòi hỏi phải thay đổi cấu trúc hệ thống hiện tại để bổ sung các API lấy dữ liệu từ cảm biến này.

Điều này dẫn đến:

1. Sửa đổi mã nguồn ở nhiều nơi, làm tăng nguy cơ sai sót.
2. Tăng độ phức tạp của hệ thống, khiến việc bảo trì trở nên khó khăn hơn.
3. Giảm tính ổn định và khả năng mở rộng lâu dài của hệ thống.

Hệ thống hiện tại không tuân thủ nguyên tắc thiết kế mở rộng nhưng không sửa đổi (**Open/Closed Principle - OCP**). Cụ thể, bất kỳ yêu cầu mở rộng nào (thêm cảm biến mới) đều đòi hỏi phải sửa đổi mã nguồn hiện có, dẫn đến nguy cơ phá vỡ tính ổn định của hệ thống.

Yêu cầu:

1. Thiết kế lại hệ thống theo nguyên tắc Open/Closed Principle (OCP), sao cho:

2. Việc tích hợp thêm cảm biến mới (cảm biến độ ẩm, áp suất, ánh sáng,...) chỉ yêu cầu mở rộng hệ thống mà không cần sửa đổi mã nguồn cũ.
3. Giải thích cách thiết kế của bạn giúp giải quyết các vấn đề nêu trên.

Gợi ý: Sử dụng **Factory Pattern** để đảm bảo hệ thống có khả năng mở rộng mà không cần sửa đổi mã nguồn hiện tại.

Bài tập 3: Trong quá trình phát triển phần mềm, bạn thường phải làm việc với các đối tượng phức tạp có nhiều thuộc tính và thành phần. Ví dụ, khi cấu hình giao tiếp UART, bạn cần thiết lập các tham số như:

1. **Baud rate,**
2. **Parity,**
3. **Stop bits,**
4. **Data bits.**

Tuy nhiên, việc khởi tạo UART theo cách thông thường gặp phải một số vấn đề nghiêm trọng như sau:

1. **Quá trình khởi tạo phức tạp**: Có quá nhiều tham số cần truyền vào, dễ gây nhầm lẫn hoặc sai sót trong quá trình cấu hình.
2. **Sự phụ thuộc giữa các thành phần**: Các tham số cấu hình có thể phụ thuộc lẫn nhau, khiến việc thay đổi một phần trở nên khó khăn mà không ảnh hưởng đến các phần khác.
3. **Khó tạo các biến thể linh hoạt**: Khi cần tạo các cấu hình UART khác nhau (ví dụ: một cấu hình cho cảm biến nhiệt độ và một cấu hình khác cho module giao tiếp không dây), bạn phải viết lại hoặc sửa đổi mã nguồn, làm giảm khả năng tái sử dụng và mở rộng.

Yêu cầu: Đề xuất và triển khai giải pháp để giải quyết các vấn đề trên, đảm bảo:

1. Quá trình khởi tạo UART trở nên đơn giản và ít lỗi hơn.
2. Việc thay đổi hoặc mở rộng cấu hình UART không làm ảnh hưởng đến các phần khác.
3. Dễ dàng tạo ra các biến thể cấu hình khác nhau mà không cần sửa đổi mã nguồn hiện tại.

4. Giải thích cách giải pháp của bạn giúp khắc phục các vấn đề đã nêu.

Gợi ý: Sử dụng Builder Pattern để quản lý quá trình khởi tạo các đối tượng phức tạp như cấu hình UART.