Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Campus: Santo Antônio de Jesus
Data: 27/07/25 | Curso: ADS
Disciplina: Padrões de Projeto
Docente: Felipe de Souza Silva
Discentes:
- Hildemar Lemos de Santana Júnior
- Kleberson de Jesus Sousa
- Thiago Sampaio Santos

Levantamento de problemas no Código Legado do sistema de monitoramento de sensores

Este relatório apresenta a análise do código legado do sistema de monitoramento de sensores, com o objetivo de identificar falhas estruturais e de design. A análise foi baseada no código-fonte fornecido e no diagrama UML atual. Os problemas destacados comprometem princípios fundamentais da engenharia de software, como coesão, baixo acoplamento e facilidade de manutenção.

1 - Levantamento de Problemas do Código Legado

1.1 - Alto acoplamento entre classes:

- A classe SensorManager instancia diretamente todas as classes concretas dos sensores, como SensorTemperatura, SensorPressao, SensorUmidade, entre outras. Isso significa que a classe depende de implementações específicas e está fortemente acoplada a elas.
- Isso dificulta a manutenção, pois qualquer alteração em uma classe concreta pode impactar diretamente SensorManager; prejudica a escalabilidade, já que a adição de um novo sensor requer modificações internas em SensorManager e viola o Princípio da Inversão de Dependência (DIP), pois a classe de alto nível (SensorManager) depende de implementações de baixo nível (sensores concretos), e não de abstrações.

1.2 - Ausência de uma interface comum entre sensores:

- Cada sensor é implementado como uma classe independente, sem herdar de uma interface ou classe abstrata comum (por exemplo, ISensor ou SensorAbstrato). Isso cria um sistema fragmentado e inconsistente.
- Isso viola o Princípio Aberto-Fechado (OCP), pois o sistema não está preparado para ser estendido sem ser modificado.

1.3 - Uso direto da classe DisplayConsole:

- A visualização dos dados capturados pelos sensores é feita diretamente através da classe DisplayConsole, sem qualquer forma de abstração ou flexibilidade.
- A exibição está fortemente acoplada ao console, dificultando a reutilização do código para outros meios de saída (como envio via rede, escrita em arquivos ou interface gráfica); viola o Princípio da Inversão de Dependência (DIP) e prejudica a extensibilidade do sistema; e dificulta a testabilidade, pois é impossível simular ou mockar a saída sem modificar a lógica de negócio.

1.4 - Baixa coesão:

- A classe SensorManager concentra várias responsabilidades: Criação dos sensores; coleta dos dados e exibição dos dados.
- Assim, ela viola o Princípio da Responsabilidade Única (SRP); torna a classe difícil de compreender, testar e manter, já que ela mistura diferentes lógicas (instanciação, processamento e apresentação), e prejudica o reaproveitamento de código, pois não é possível isolar funcionalidades específicas com clareza.

1.5 - Falta de aplicação de princípios SOLID:

- O código legado, como um todo, não segue os princípios SOLID, que são fundamentais para a construção de software flexível e de fácil manutenção. A classe SensorManager viola o DIP (Princípio da Inversão de Dependência) ao acumular múltiplas funções (instanciação, processamento e exibição). Viola o OCP (Princípio Aberto-Fechado), pois, o sistema não está preparado para ser estendido (como adicionar novos sensores) sem modificações diretas no código existente. E viola também o SRP (Princípio da Responsabilidade Única), porque não há inversão de dependência. As dependências são criadas internamente, em vez de serem injetadas a partir de abstrações.

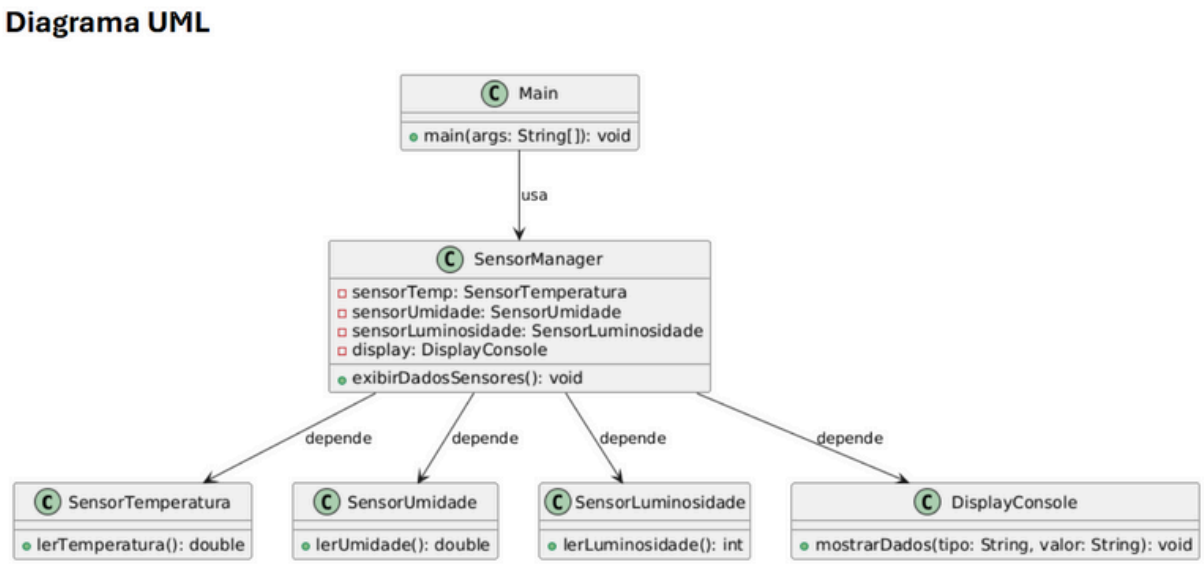

Figura 1 – Diagrama atual do sistema antes da refatoração.

2 - Aplicação de Padrões de Criação

Foi aplicado o padrão de projeto Factory Method com o objetivo de desacoplar a criação dos sensores do restante do sistema. Cada tipo de sensor (temperatura, umidade, luminosidade) possui agora sua própria fábrica dedicada, que estende a classe abstrata SensorFactory e encapsula a lógica de instanciamento.

Essa abordagem facilita a extensão do sistema: para incluir um novo sensor, basta criar uma nova subclasse de SensorFactory, sem a necessidade de modificar o código do gerenciador (SensorManager) ou da interface de exibição (DisplayConsole). Isso está de acordo com o Princípio Aberto-Fechado (OCP).

Já o padrão Singleton foi aplicado na classe SensorManager, garantindo que haja apenas uma instância centralizada para o gerenciamento dos sensores. Isso é útil em sistemas embarcados ou distribuídos, onde múltiplas instâncias poderiam causar conflitos de estado ou duplicações de leitura. A implementação do Singleton também contribui para a facilidade de acesso compartilhado e para o controle centralizado da aplicação.

3 - Aplicação dos Padrões Adapter e Decorator

O padrão Adapter foi aplicado com o objetivo de tornar os sensores legados compatíveis com o novo sistema baseado na interface Sensor. Cada classe de sensor legado (SensorTemperatura, SensorUmidade, SensorLuminosidade) foi encapsulada por um adaptador correspondente (SensorTemperaturaAdapter, etc.). Assim, o sistema pode usar sensores antigos sem alterar sua implementação original — respeitando o Princípio Aberto-Fechado (OCP) e promovendo a reutilização de código legado com baixo impacto.

Já o padrão Decorator foi utilizado para adicionar comportamentos extras de forma flexível e modular, sem modificar as classes concretas dos sensores. Foram implementados três decoradores:

- LogDecorator: registra automaticamente cada leitura realizada, facilitando a auditoria e o rastreamento de dados.
- AlertaDecorator: verifica condições críticas (como temperatura > 50 °C) e dispara alertas em tempo real.
- FormatadorDecorator (caso você implemente): padroniza a saída dos dados para diferentes formatos (ex: JSON, texto simples, etc.).

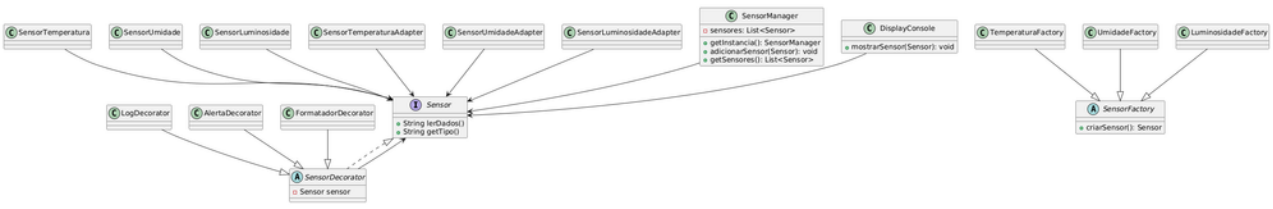

Figura 2 – Diagrama do sistema depois da refatoração.