Sistema Modular para Gerenciamento de Dispositivos Inteligentes

Kleberson de Jesus Sousa e Thiago Sampaio Santos

Santo Antônio de Jesus, Bahia - 2025

1. Descrição do Sistema

O Sistema de Gerenciamento de Dispositivos Inteligentes é uma aplicação modular desenvolvida em Java 17, projetada para controlar e monitorar dispositivos como sensores e atuadores. Ele suporta o cadastro e gerenciamento de dispositivos, o monitoramento em tempo real de eventos e a execução de ações automáticas de acordo com condições pré-definidas.

Seu funcionamento se baseia em três pilares principais:
1. Gerenciamento Centralizado — através de um gerenciador único (Singleton), é possível coordenar todos os dispositivos e eventos.
2. Modularidade — o sistema é construído sobre interfaces e padrões de projeto, permitindo que novos dispositivos, comandos e estratégias sejam adicionados sem modificar código existente.
3. Extensibilidade — a estrutura permite futuras integrações com IoT, bancos de dados e APIs de terceiros.

A aplicação simula, por exemplo, um cenário onde sensores de temperatura enviam eventos, e uma estratégia de resposta pode acionar automaticamente um ventilador caso a temperatura ultrapasse um limite configurado.

2. Justificativas dos Padrões Aplicados

O projeto adota múltiplos padrões de projeto GoF, cada um com um papel específico:

- Factory Method - Facilita a criação de dispositivos (sensores e atuadores) sem acoplar o código às classes concretas.
- Singleton: Garante que componentes centrais, como o Gerenciador do Sistema e o Gerenciador de Eventos, tenham uma única instância global, evitando conflitos de estado.
- Observer: Permite que eventos sejam propagados de forma assíncrona para ouvintes interessados, reduzindo o acoplamento entre emissores e receptores e facilitando a adição de novos comportamentos.
- Comman : Encapsula ações (como ligar um ventilador) em objetos, permitindo o agendamento, repetição ou cancelamento de comandos.
- Strategy : Facilita a troca de algoritmos de resposta a eventos, sem alterar o código cliente, tornando possível alterar o comportamento do sistema em tempo de execução.
- Decorator + Composite : Permitem a composição de elementos visuais e aplicação de temas dinâmicos, mantendo a flexibilidade de personalização e organização hierárquica.

Esses padrões foram escolhidos porque aumentam a coesão, reduzem o acoplamento e tornam o sistema mais fácil de manter e evoluir.

3. Análise de Aderência aos Princípios SOLID

O sistema demonstra aderência aos princípios SOLID da seguinte forma:

- SRP (Responsabilidade Única): Cada classe possui um papel bem definido (ex.: `FabricaDeDispositivos` apenas cria dispositivos; `ComandoLigarVentilador` apenas encapsula o comando de ligar ventilador).
- OCP (Aberto/Fechado): É possível adicionar novos tipos de dispositivos, comandos e estratégias sem modificar o código existente.
- LSP (Substituição de Liskov): As subclasses de `Dispositivo` podem ser usadas de forma intercambiável com sua superclasse sem alterar o comportamento esperado.
- ISP (Segregação de Interfaces): Interfaces específicas (como `Comando` e `OuvinteDeEvento`) evitam que classes sejam obrigadas a implementar métodos irrelevantes.
- **DIP (Inversão de Dependência): Classes dependem de abstrações (interfaces) e não de implementações concretas, permitindo a injeção de dependências.

Essa aplicação prática dos princípios SOLID garante que o sistema seja flexível, testável e escalável.

4. Possíveis Extensões Futuras

O projeto possui diversas oportunidades de expansão:

- Novos Dispositivos: Adicionar dispositivos como lâmpadas inteligentes, e fechaduras eletrônicas .
- Integração com APIs Externas: Conectar-se a serviços de IoT para controle remoto pela internet.
- Persistência de Dados: Uso de banco de dados relacional ou NoSQL para salvar o histórico de eventos e estados dos dispositivos.
- Interface Gráfica: Implementar GUI usando JavaFX ou frameworks web para tornar a interação mais intuitiva.
- Configuração Dinâmica de Estratégias: Permitir que usuários configurem estratégias e regras via interface ou arquivos JSON/YAML.

Essas melhorias manteriam a arquitetura modular e aumentariam a usabilidade e aplicabilidade do sistema em cenários reais.

5. Tabela de Trade-offs entre Padrões Usados

Padrão	**Vantagens**	**Desvantagens**
Factory Method	Desacopla criação de objetos e facilita extensão	Pode aumentar o número de classes
Singleton	Controle centralizado e único ponto de acesso	Pode dificultar testes unitários
Observer	Alta flexibilidade e baixo acoplamento	Possível complexidade no rastreamento de eventos
Command	Facilita o encapsulamento e agendamento de ações	Mais classes e interfaces para gerenciar
Strategy	Facilita troca de algoritmos em tempo de execução	Pode gerar excesso de classes para estratégias simples
Decorator	Permite adicionar funcionalidades dinamicamente	Pode aumentar a complexidade de depuração
Composite	Organiza hierarquia de objetos complexos	Complexidade extra na implementação