DESIGN PATTERNS

PADRÕES ESTRUTURAIS

Prof. Sandro Ramos

OVERVIEW

- Padrões estruturais
- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

PADRÃO ESTRUTURAL

São descritas como soluções para problemas relacionados à composição de classes e objetos para formar estruturas maiores e mais flexíveis.

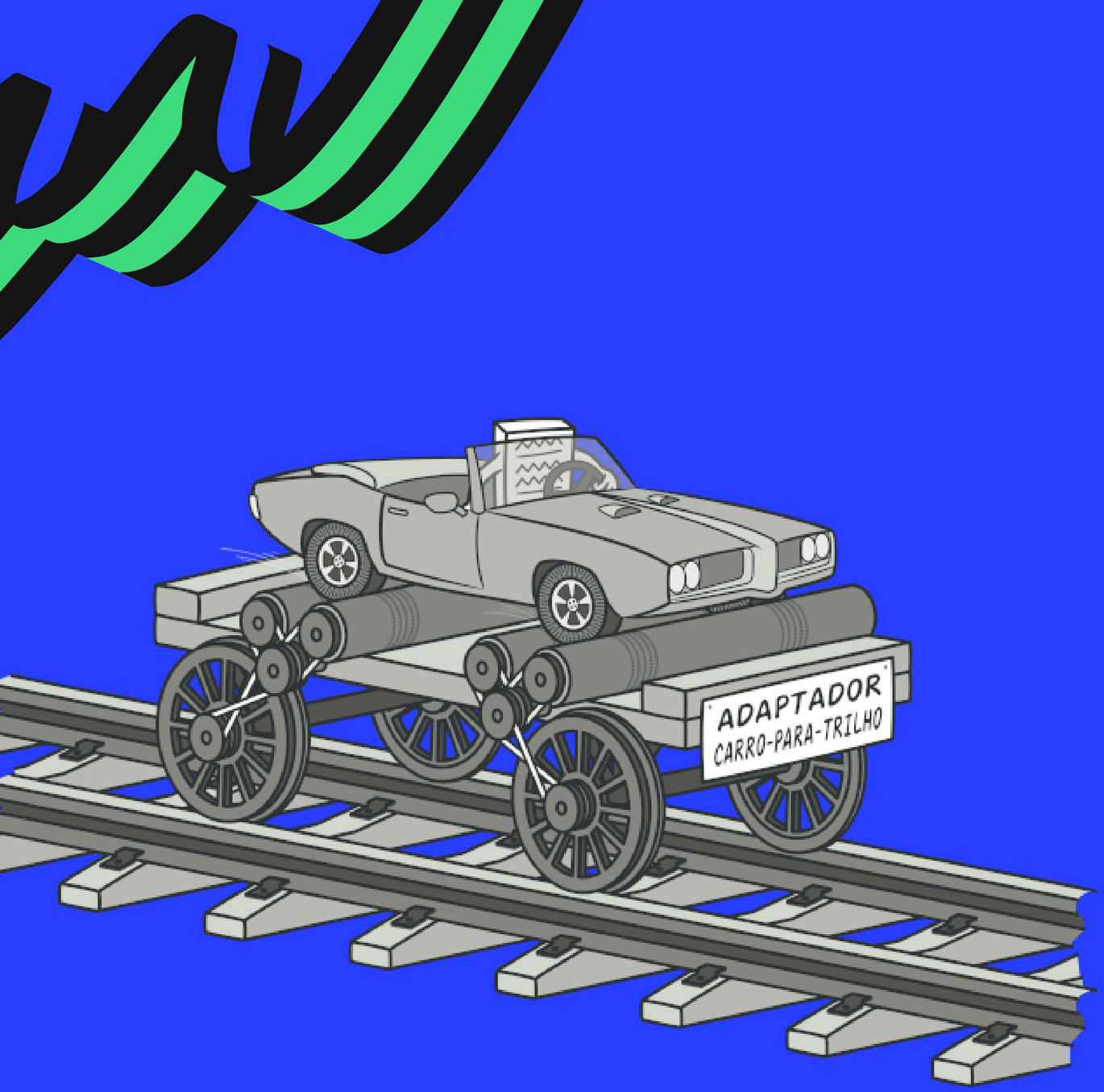
Esses padrões ajudam a organizar e otimizar o design de sistemas de software, facilitando como os componentes interagem e se conectam.

Principais benefícios:

- Flexibilidade na composição de objetos
- Desacoplamento
- Manutenção e extensão
- Simplificação de complexidade

"Structural patterns deal with object composition, creating relationships between objects to form larger structures. These patterns help ensure that if one part of a system changes, the entire system does not need to change. They simplify the design by decoupling components, making the system more flexible and easier to maintain." - Gamma et al., 1995, p. 25-26

ADAPTER



Permite que interfaces incompatíveis trabalhem juntas. Ele converte a interface de uma classe em outra interface esperada pelos clientes.

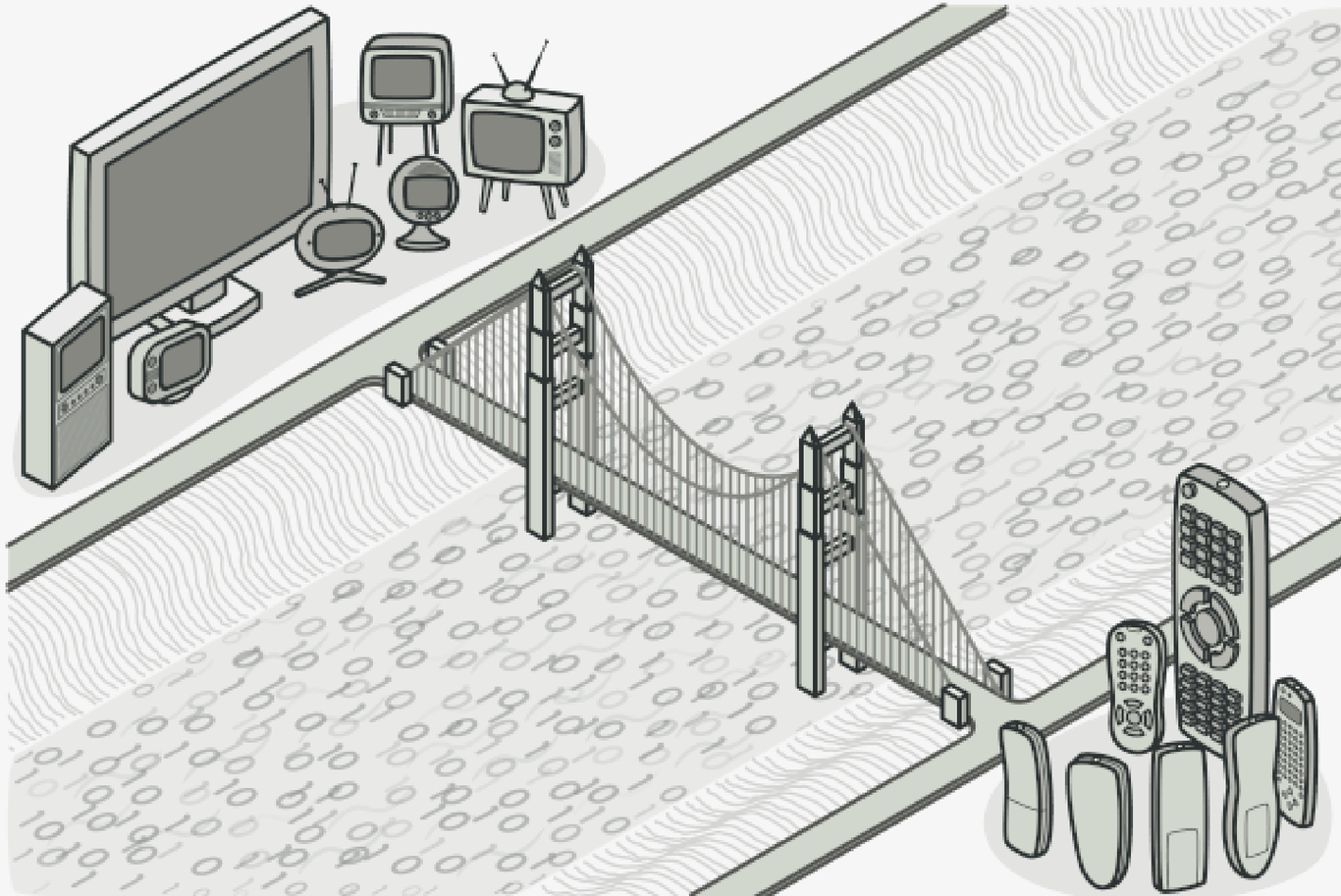
Assim, o Adapter atua como um intermediário que traduz as chamadas feitas pelo cliente para a interface da classe que não é compatível.

Usado quando é necessário integrar sistemas legados com novos sistemas ou quando se deseja adaptar uma API para uma nova interface.

BRIDGE

Desacopla uma abstração da sua implementação, permitindo que ambas evoluam independentemente.

É usado quando você quer separar uma abstração (a ideia principal ou conceito) de sua implementação concreta (a forma como o conceito é realizado). Isso é útil para evitar que mudanças em uma parte da estrutura (como uma classe) causem impacto na outra parte.

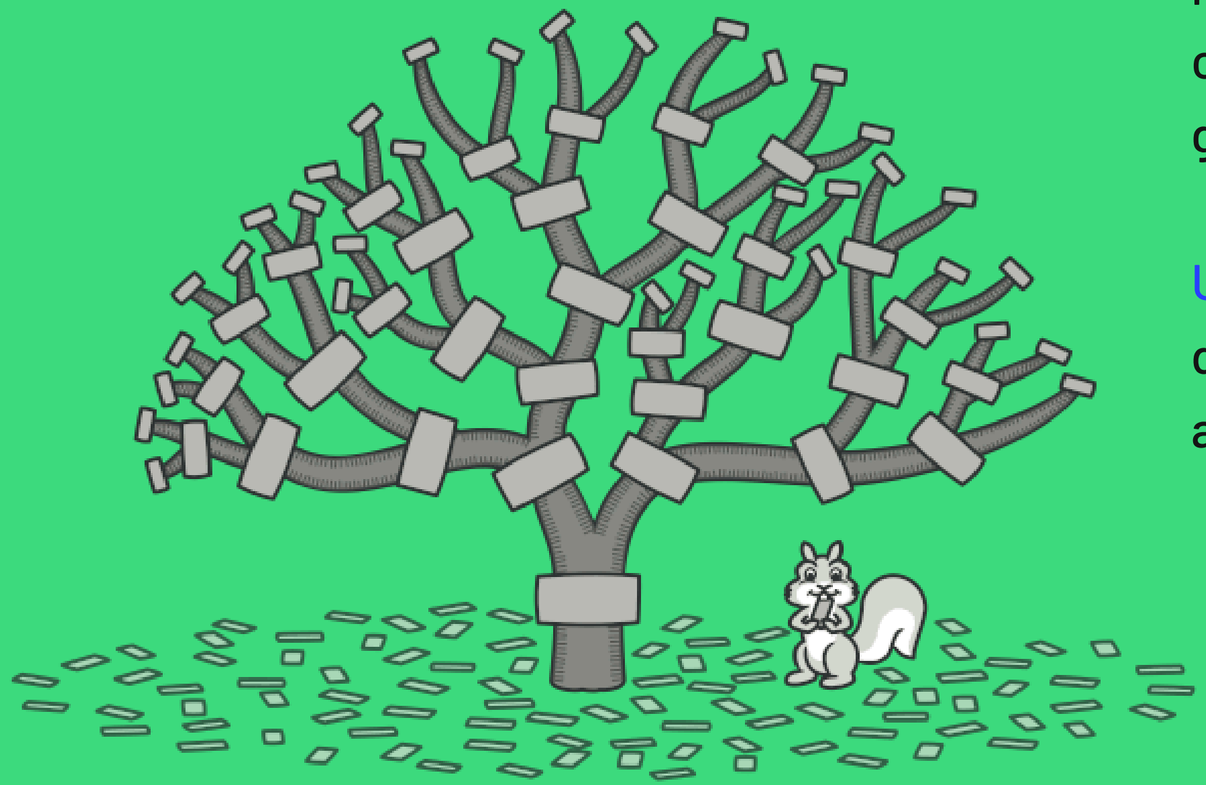


COMPOSITE

Permite que você trate objetos individuais e composições de objetos de forma uniforme.

Isso é útil quando você tem uma estrutura hierárquica de objetos, como uma árvore, onde cada nó pode ser um objeto simples ou um grupo de objetos.

Usado quando precisar representar hierarquias de objetos, como estruturas de diretórios de arquivos ou interfaces gráficas complexas.



DECORATOR

Permite adicionar responsabilidades a um objeto dinamicamente, sem alterar o código da classe original. Ele usa um conjunto de classes de decoração que envolvem o objeto original, adicionando funcionalidades adicionais. Utilizado para adicionar funcionalidades a objetos de maneira flexível e extensível, como adicionar estilos de janela em um sistema de GUI.

PROBLEMAS!!

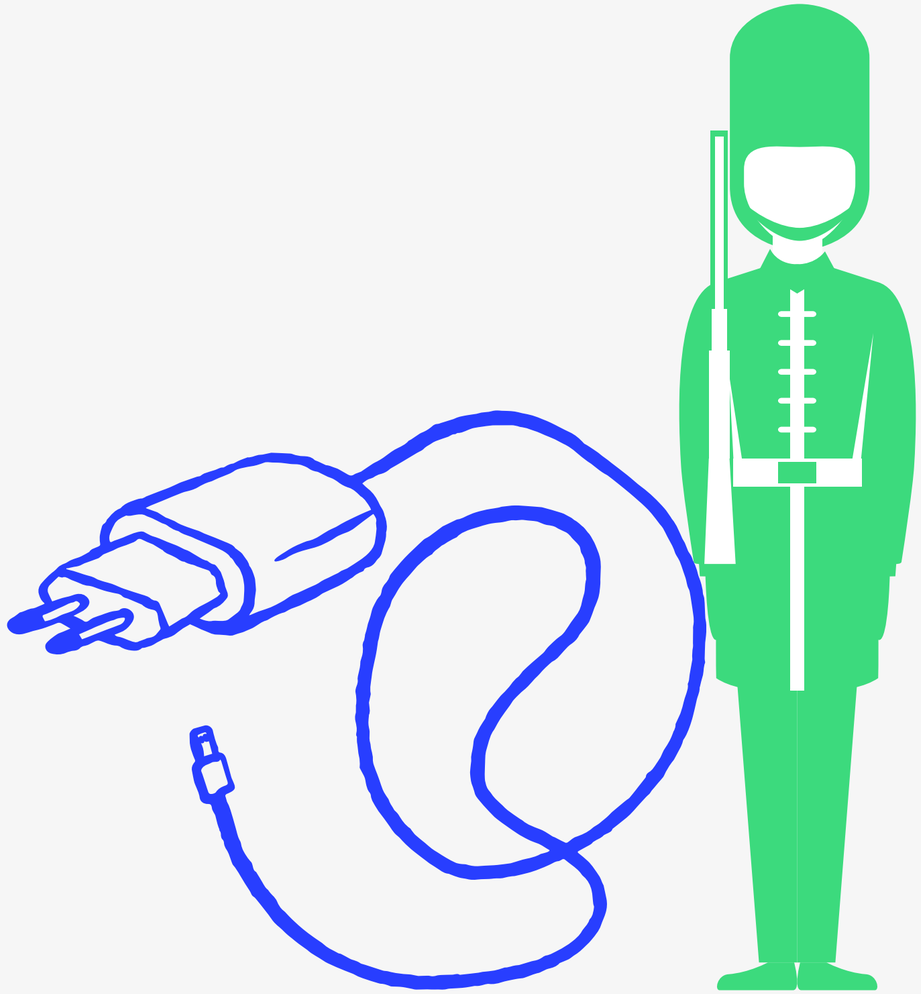
1º PROBLEMA

Desejamos criar um sistema que precisa lidar com a estrutura de um sistema de arquivos, onde há pastas que podem conter arquivos e outras pastas.

Cada pasta pode ter um número arbitrário de subpastas e arquivos, mas tanto os arquivos quanto as pastas podem ser tratados como "componentes" do sistema de arquivos, permitindo operações como exibir o conteúdo ou calcular o tamanho total.

Repo exemplo.

2º PROBLEMA



Imagine que você está fazendo um intercambio para o Reino Unido, chegando lá com seu laptop, você descobre que o padrão de tomada é diferente.

Seu carregador foi projetado para funcionar com uma tomada de dois pinos (padrão BR), mas lá são utilizadas tomadas de três pinos (padrão UK).

Para resolver esse problema sem a necessidade de modificar o design do carregador ou da tomada, você pode usar um adaptador de tomada que faz a conversão entre esses dois padrões.

Repo exemplo.

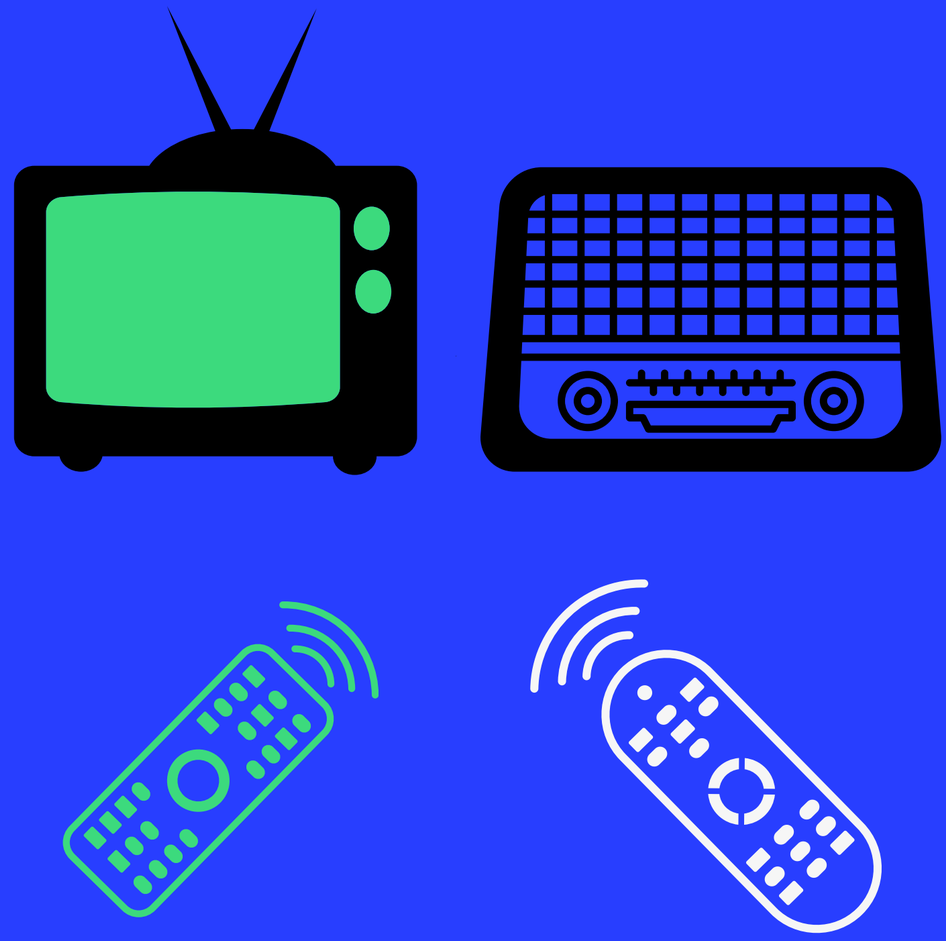
3º PROBLEMA

Imagine que você está desenvolvendo um sistema que controla dispositivos eletrônicos como TVs e rádios.

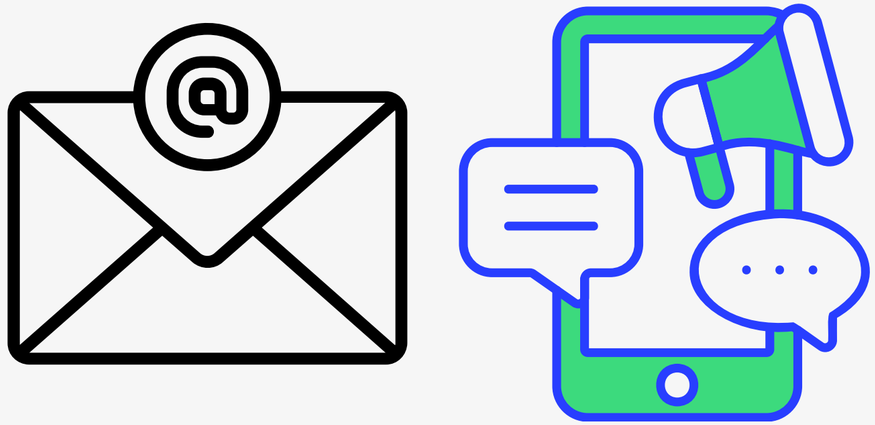
Você precisa criar diferentes tipos de controles remotos (por exemplo, controles básicos e avançados) para esses dispositivos.

Uma abordagem simples seria criar algo específico para cada tipo de controle e dispositivo (por exemplo, Controle Remoto TV, Controle Remoto Rádio), mas isso levaria a uma explosão de implementações conforme novos dispositivos e novos tipos de controles surgem.

Repo exemplo.



4º PROBLEMA



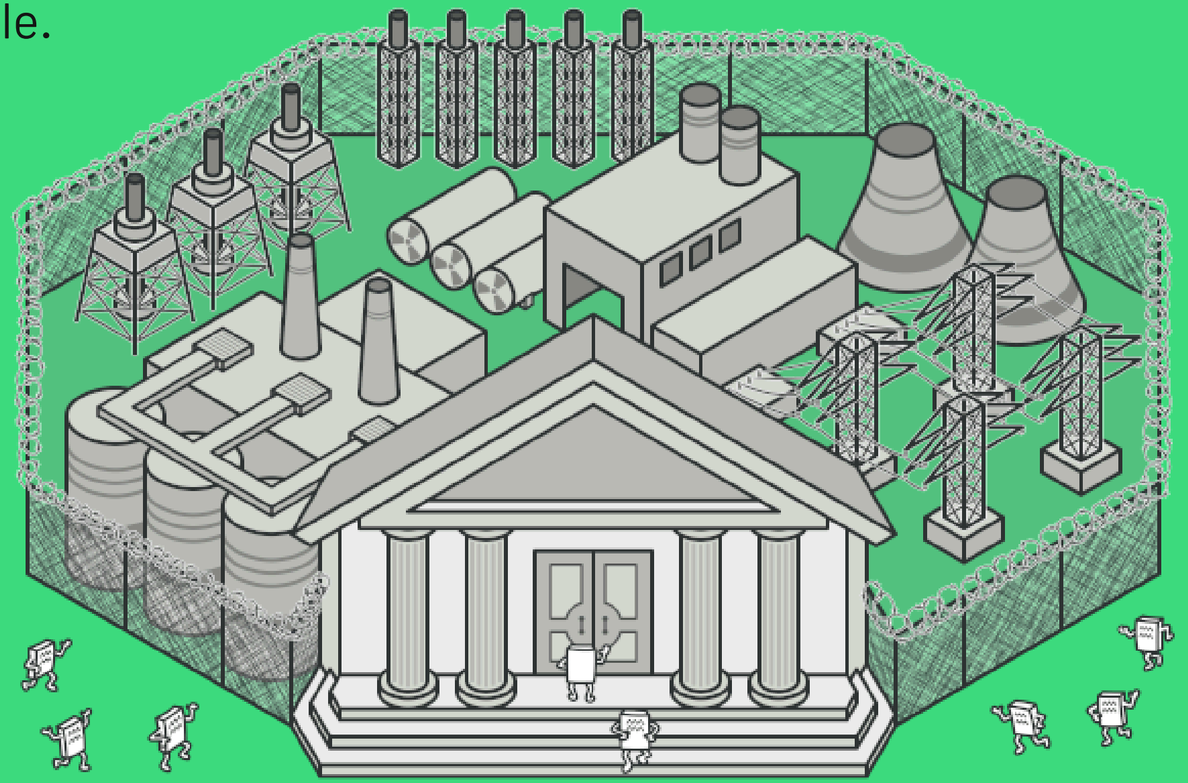
Imagine que você tem um sistema de notificações, a notificação básica envia mensagens via e-mail.

No entanto, você deseja adicionar a capacidade de enviar mensagens via SMS e registrar as notificações em um log.

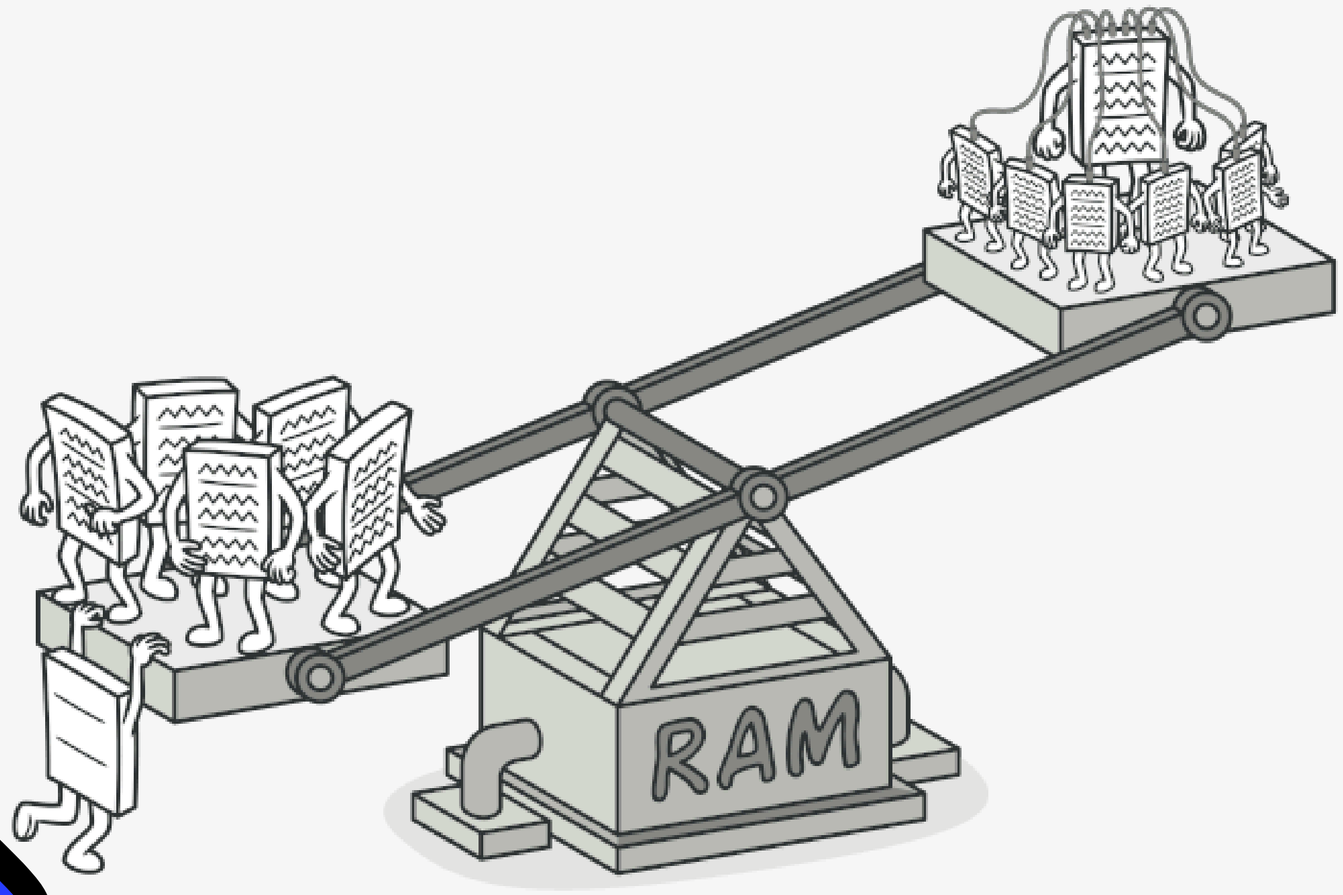
Repo de exemplo.

FACADE

Fornece uma interface simplificada para um conjunto de interfaces em um subsistema complexo. Em outras palavras, o Facade atua como uma "fachada" que oculta a complexidade de um sistema, oferecendo uma maneira mais fácil e direta para os clientes interagirem com ele.



FLYWEIGHT



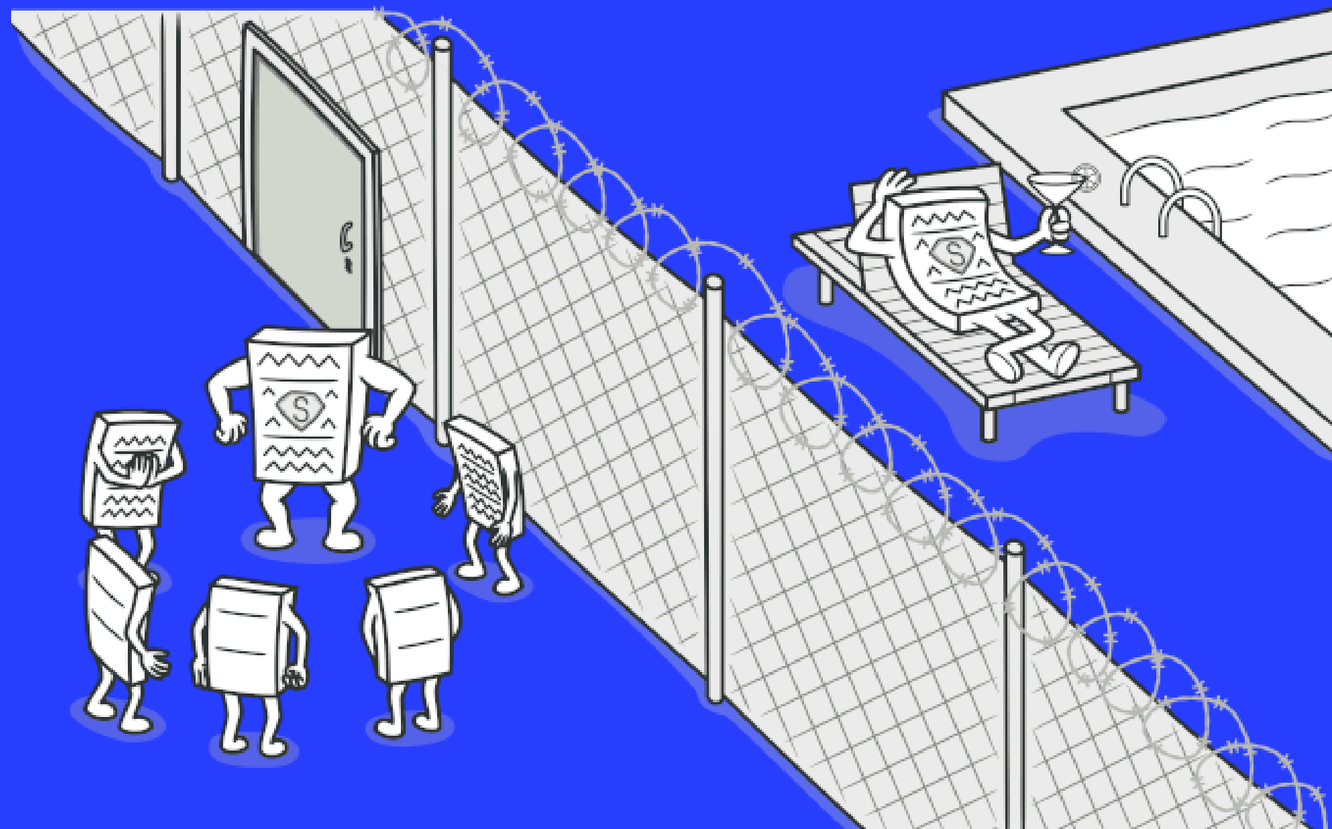
Visa reduzir o uso de memória e melhorar o desempenho ao compartilhar objetos semelhantes em vez de criar novos objetos para cada instância.

Ele é especialmente útil quando você precisa lidar com excesso de objetos que possuem características semelhantes.

PROXY

Fornece um substituto ou intermediário para outro objeto. Ele age como um "representante" do objeto real, controlando o acesso a ele e podendo adicionar funcionalidades extras, como controle de acesso, atraso na criação ou manipulação adicional.

Usado para adicionar funcionalidades como controle de acesso, cache, ou outras operações intermediárias.

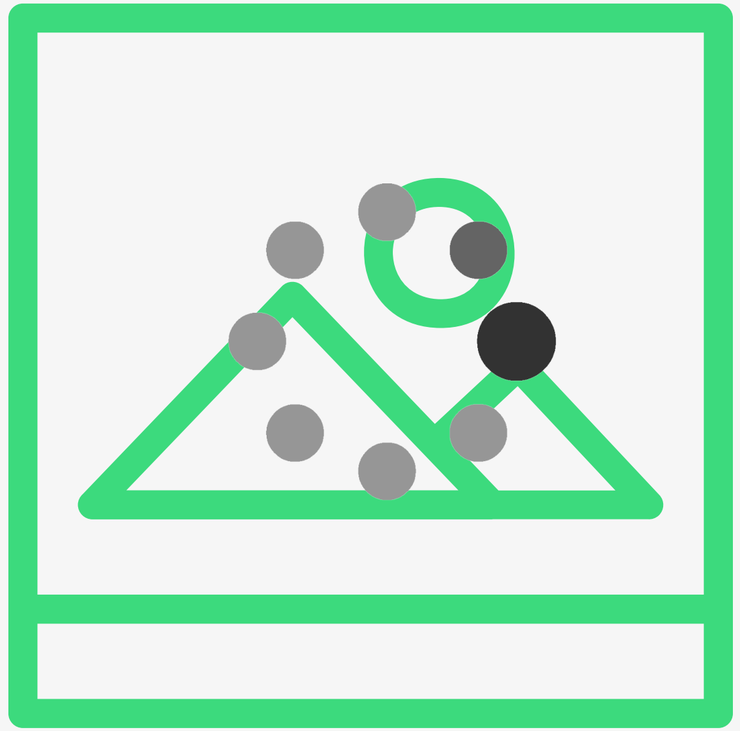


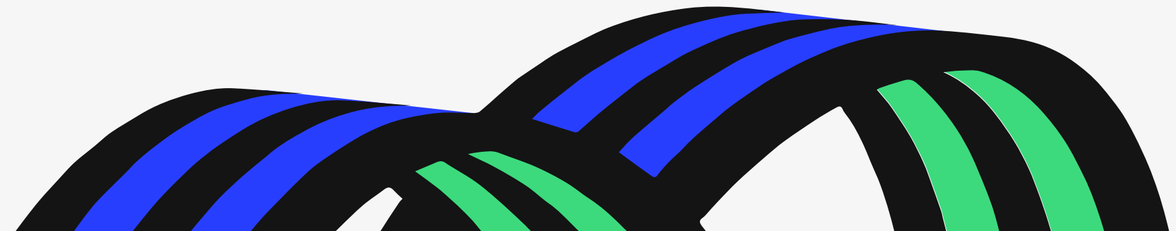
5º PROBLEMA

Imagine que você está desenvolvendo um sistema de gerenciamento de imagens que precisa exibir várias imagens de alta resolução.

Carregar todas as imagens de uma vez pode consumir muita memória e causar lentidão.

Repo exemplo.





6º PROBLEMA

Imagine que você está criando um **Age of Empiers** com inúmeros personagens no campo de batalha.

Cada personagem tem um tipo (por exemplo, arqueiro, cavaleiro, monge) e atributos específicos (força, velocidade, etc.).

No entanto, muitos personagens compartilham o mesmo tipo, logo, criar uma nova instância de cada tipo para cada personagem resultaria em desperdício de memória.

Repo exemplo.

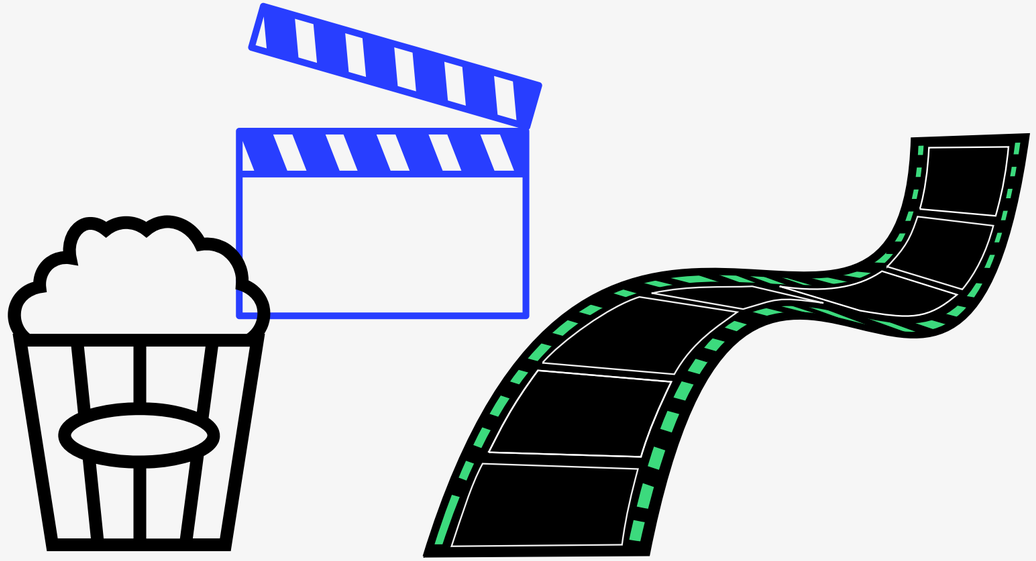
7º PROBLEMA

Imagine que estamos desenvolvendo um sistema de gerenciamento de cinema.

O processo de reserva de um ingresso envolve múltiplos passos, como escolher o filme, selecionar assentos, fazer o pagamento e enviar um e-mail de confirmação.

Caso o cliente tenha de lidar com cada uma dessas operações, iremos aumentar a complexidade do código e abrir margem para erros.

Repo exemplo.



Referências

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

XP EDUCAÇÃO. Padrões de projeto. Disponível em: https://blog.xpeducacao.com.br/padroes-de-projeto/. Acesso em: 08 set. 2024.

ROBERTO, Jones. Design patterns – Parte 2. Medium, 2023. Disponível em: https://medium.com/@jonesroberto/desing-patterns-parte-2-2a61878846d. Acesso em: 10 set. 2024.

REFACTORIG GURU. Padrões de projeto de software. Disponível em: https://refactoring.guru. Acesso em: 08 set. 2024.