Padrões Estruturais GoF

Os **Padrões Estruturais** do **Gang of Four (GoF)** são usados para organizar classes e objetos, formando estruturas maiores e mais flexíveis. Eles se concentram na composição de objetos para formar sistemas complexos, promovendo reutilização e flexibilidade sem alterar diretamente as classes. Incluem os seguintes padrões:

---

1. **Adapter**

O padrão Adapter define uma estrutura de adaptadores que convertem uma interface comum em uma interface fornecida por uma implementação específica, tipicamente fornecida por terceiros

- **Objetivo:** Converter a interface de uma classe em outra que os clientes esperam, permitindo a compatibilidade entre classes com interfaces diferentes.
- **Aplicação:** Útil para integrar sistemas legados ou componentes de terceiros que possuem interfaces incompatíveis.

No padrão Adapter, Target define uma interface genérica comum que os clientes deverão utilizar. Adaptee corresponde a uma implementação específica de uma interface definida pelo fornecedor do componente. O participante Adapter é responsável por adaptar as chamadas à interface genérica (Target) para a API específica (Adaptee). Portanto, as operações definidas em cada Adapter devem possuir a mesma assinatura daquelas definidas no participante Target.

- **Exemplo:** Adaptadores de tomada para conectar dispositivos com diferentes padrões.

---

2. **Bridge**

- **Objetivo:** Desacoplar a abstração de sua implementação para que ambas possam evoluir independentemente.
- **Aplicação:** Evita a proliferação de subclasses quando múltiplas combinações de abstrações e implementações são necessárias.

O padrão Bridge permite evitar a criação de múltiplas subclasses para cada combinação de abstração e implementação, desacoplando esses dois aspectos e promovendo um design mais flexível e extensível. Isso reduz a complexidade do sistema, facilitando a manutenção e evolução ao longo do tempo.

- **Exemplo:** Componentes gráficos como botões que podem ser renderizados em diferentes sistemas operacionais.

---

3. **Composite**

- **Objetivo:** Compor objetos em estruturas de árvore para representar hierarquias parte-todo, tratando objetos individuais e composições de maneira uniforme.

- **Aplicação:** Útil para representar hierarquias como pastas e arquivos em um sistema de diretórios.
- **Exemplo:** Estrutura de menus em um sistema de navegação.

---

4. **Decorator**
    - **Objetivo:** Adicionar responsabilidades a objetos de forma dinâmica, sem usar herança, permitindo combinações flexíveis de comportamentos.

Adicionar funcionalidades a uma classe sem utilizar subclasses, mas por meio de uma estrutura de composição dinâmica e flexível.

- **Aplicação:** Quando queremos estender funcionalidades de objetos de forma dinâmica e transparente.
- **Exemplo:** Adicionar criptografia e compressão a fluxos de dados.

---

5. **Facade**
    - **Objetivo:** Fornecer uma interface simplificada para um subsistema complexo, reduzindo o acoplamento entre o cliente e o sistema.
    - **Aplicação:** Ideal para sistemas que expõem muitas interfaces complexas para o cliente.
    - **Exemplo:** Interfaces de alto nível para APIs de serviços externos.

---

6. **Flyweight**
    - **Objetivo:** Minimizar o uso de memória compartilhando dados comuns entre muitos objetos semelhantes.
    - **Aplicação:** Útil quando se precisa instanciar muitos objetos com dados repetidos.
    - **Exemplo:** Representação de caracteres em editores de texto.

---

7. **Proxy**
    - **Objetivo:** Controlar o acesso a objetos, adicionando uma camada de controle que pode incluir autenticação, carregamento preguiçoso ou cache.
    - **Aplicação:** Quando precisamos controlar o acesso a objetos que demandam muitos recursos ou que estão em locais remotos.
    - **Exemplo:** Conexões remotas ou objetos que exigem controle de acesso.

Conclusão:

Os padrões estruturais são fundamentais para a construção de sistemas flexíveis e escaláveis, permitindo a reutilização de componentes, a separação de preocupações e a redução do acoplamento entre módulos. Cada padrão aborda um aspecto específico da organização de classes e objetos, oferecendo soluções para problemas comuns na construção de sistemas complexos.