Tema 3

Módulo 1

**Adapter (Adaptador)**

**Conceito:**  
Permite que classes com interfaces incompatíveis trabalhem juntas, convertendo a interface de uma classe para outra que o cliente espera.

(O propósito do Adapter é facilitar a integração e reutilização de classes existentes em novos contextos, especialmente em sistemas legados, proporcionando flexibilidade e compatibilidade sem a necessidade de modificar as classes originais.)

**Vantagens:**

* Facilita a integração de classes que, originalmente, não poderiam colaborar.
* Separa as responsabilidades, promovendo a reutilização de código existente.

**Desvantagens:**

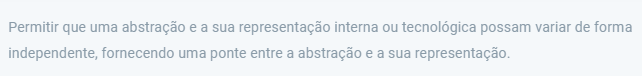
* Pode introduzir uma camada extra de indireção, aumentando a complexidade.
* Em sistemas muito grandes, o uso excessivo pode dificultar o rastreamento do fluxo de execução.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

**Bridge (Ponte)**

**Conceito:**  
Separa a abstração de sua implementação, permitindo que ambas evoluam de forma independente.



(Resolve o problema da proliferação de subclasses que ocorre quando é necessário combinar múltiplas abstrações com múltiplas implementações.)

A ideia proposta pelo padrão é separar a hierarquia de abstrações da hierarquia de implementações, fazendo com que a implementação das operações na abstração (participante Abstraction) seja responsável por delegar a execução para a implementação específica que estiver conectada à abstração (participante ConcreteImplementor).

**Vantagens:**

* Aumenta a flexibilidade, possibilitando a variação independente de abstrações e implementações.
* Facilita a manutenção e a extensão do sistema.

**Desvantagens:**

* Pode introduzir complexidade adicional ao sistema devido à criação de múltiplas camadas.
* Requer um bom planejamento para que a separação seja efetiva.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

**Decorator (Decorador)**

**Conceito:**  
Adiciona dinamicamente novas responsabilidades a um objeto de modo dinâmico e transparente, sem a necessidade de usar subclasses, oferecendo uma alternativa flexível à herança.

**Vantagens:**

* Permite estender funcionalidades de forma dinâmica e flexível.
* Reduz o número de subclasses necessárias para agregar comportamentos.

**Desvantagens:**

* Pode resultar em muitos objetos pequenos e encapsulados, o que dificulta a depuração.
* A composição de múltiplos decoradores pode complicar a compreensão do fluxo de execução.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

Todo Decorator recebe, no seu construtor, um parâmetro correspondente ao participante Component do padrão.

**Composite (Composto)**

**Conceito:**  
Organiza objetos em estruturas hierárquicas (em forma de árvore) para representar relações parte-todo, permitindo tratar individualmente objetos simples e composições de forma uniforme.

**Vantagens:**

* Simplifica o tratamento de estruturas complexas, permitindo operações uniformes em objetos individuais e composições.
* Facilita a adição de novos tipos de componentes sem grandes mudanças na estrutura.

**Desvantagens:**

* Pode levar a uma estrutura excessivamente genérica, dificultando a aplicação de restrições específicas a certos componentes.
* A complexidade do gerenciamento da hierarquia pode aumentar em sistemas grandes.

**Facade (Fachada)**

**Conceito:**  
Fornece uma interface simplificada para um conjunto complexo de classes ou subsistemas, facilitando o uso do sistema.

**Vantagens:**

* Reduz a complexidade do sistema para o cliente, escondendo detalhes internos.
* Promove o desacoplamento entre o cliente e o subsistema.

**Desvantagens:**

* Pode limitar o acesso a funcionalidades mais específicas do subsistema.
* Se sobrecarregada, a fachada pode se transformar em um "deus objeto", concentrando muitas responsabilidades.

**Flyweight (Flyweight ou Peso-Mosca)**

**Conceito:**  
Minimiza o uso de memória compartilhando partes comuns dos estados entre muitos objetos, mantendo separados os estados intrínsecos (compartilhados) e extrínsecos (variáveis).

**Vantagens:**

* Reduz significativamente o consumo de memória e melhora o desempenho em sistemas com grande número de objetos.
* Permite a criação eficiente de objetos em ambientes com recursos limitados.

**Desvantagens:**

* A separação entre estados intrínsecos e extrínsecos pode aumentar a complexidade do código.
* A correta gestão do compartilhamento pode ser desafiadora, exigindo um controle cuidadoso.

**Proxy (Proxy)**

**Conceito:**  
Fornece um substituto ou representante de um objeto para controlar o acesso a ele, podendo adicionar funcionalidades adicionais como cache, controle de acesso ou registro de log.

**Vantagens:**

* Permite controle sobre o acesso ao objeto real, podendo implementar funcionalidades adicionais (como lazy loading).
* Pode melhorar a segurança e o desempenho ao gerenciar recursos de forma inteligente.

**Desvantagens:**

* Introduz uma camada extra de indireção, o que pode afetar o desempenho se não for bem implementado.
* A complexidade do sistema pode aumentar, especialmente se múltiplos proxies forem utilizados.