**Structural design patterns**

**Structural design patterns:** e uma padrão que se preocupados com e como as classes de objetos podem ser compostos, para formar estruturas maiores. simplificando a estrutura identificando os relacionamentos.

Esses padrões se concentram em como as classes herdam umas das outras e como elas são compostas de outras classes.

É uma questão de otimizar, salvar processos de herança. Por exemplo, os objetos podem ser incluídos em uma estrutura de árvore, que usa a mesma interface, ou as mesmas propriedades gerais, podem ser movidas para um único objeto, que é compartilhado por todos os outros objetos.

**Tipos de padrões de projeto estrutural**

Existem 7 tipos de padrões de projeto estruturais a seguir.

* **Adapter Pattern:**
* Adaptar uma interface em outra de acordo com a expectativa do cliente.
* "converte a interface de uma classe em outra interface que um cliente deseja".
* Em outras palavras, fornecer a interface de acordo com o requisito do cliente ao usar os serviços de uma classe com uma interface diferente.
* O Padrão Adaptador também é conhecido como Wrapper.
* **Vantagem do Adapter Pattern**
* Ele permite que dois ou mais objetos anteriormente incompatíveis interajam.
* Permite a reutilização de funcionalidades existentes.
* **Bridge Pattern:**
* Separando a abstração (interface) da implementação.
* "desacople a abstração funcional da implementação para que as duas possam variar independentemente".
* O Bridge Pattern também é conhecido como Handle or Body.
* **Vantagem do Bridge Pattern**
* Ele permite a separação da implementação da interface.
* Melhora a extensibilidade.
* Permite ocultar detalhes de implementação do cliente.
* **Composite Pattern:**
* Permitindo que os clientes operem na hierarquia de objetos.
* "permite que os clientes operem de maneira genérica em objetos que podem ou não representar uma hierarquia de objetos".
* **Vantagem do Composite Pattern**
* Ele define hierarquias de classes que contêm objetos primitivos e complexos.
* Torna mais fácil para você adicionar novos tipos de componentes.
* Ele fornece flexibilidade de estrutura com classe ou interface gerenciável.
* **Decorator Pattern:**
* Adicionando funcionalidade a um objeto dinamicamente.
* "anexe responsabilidades adicionais flexíveis a um objeto dinamicamente".
* Em outras palavras, o padrão Decorator usa composição em vez de herança para estender a funcionalidade de um objeto em tempo de execução.
* O Decorator Pattern também é conhecido como Wrapper.
* **Vantagem do Decorator Pattern**
* Ele fornece maior flexibilidade do que a herança estática.
* Ele aumenta a extensibilidade do objeto, porque as alterações são feitas pela codificação de novas classes.
* Ele simplifica a codificação, permitindo que você desenvolva uma série de funcionalidades de classes direcionadas em vez de codificar todo o comportamento no objeto.
* **Facade Pattern:**
* Fornecendo uma interface para um conjunto de interfaces.
* "apenas fornece uma interface unificada e simplificada para um conjunto de interfaces em um subsistema, portanto, oculta as complexidades do subsistema do cliente".
* Em outras palavras, o Facade Pattern descreve uma interface de nível superior que torna o subsistema mais fácil de usar.
* Praticamente, toda Abstract Factory é um tipo de Fachada.
* **Vantagem do Facade Pattern**
* Ele protege os clientes das complexidades dos componentes do subsistema.
* Promove o baixo acoplamento entre os subsistemas e seus clientes.
* **Flyweight Pattern:**
  + Reutilizando um objeto compartilhando-o.
  + "para reutilizar tipos semelhantes de objetos já existentes, armazenando-os e criando um novo objeto quando nenhum objeto correspondente for encontrado" .
  + **Vantagem do Flyweight Pattern**
  + Reduz o número de objetos.
  + Reduz a quantidade de memória e dispositivos de armazenamento necessários se os objetos persistirem
* **proxy Pattern:**
  + Simplesmente, proxy significa um objeto que representa outro objeto.
  + De acordo com o GoF, um **proxy Pattern** "fornece o controle para acessar o objeto original".
  + Assim, podemos realizar muitas operações como ocultar as informações do objeto original, carregamento sob demanda etc.
  + O **proxy Pattern** também é conhecido como Surrogate or Placeholder.
  + Vantagem do Padrão Proxy
  + Ele fornece a proteção ao objeto original do mundo exterior.