**Padrão de Projeto**

* **O que é?**

Em engenharia de software, um **padrão de projeto** é uma solução geral reutilizável para um problema que ocorre com frequência dentro de um determinado contexto no projeto de software.

* **O que define?**

Seu nome, o problema, quando aplicar esta solução e suas consequências.

* **Quais os objetivos?**
* Facilitar a reutilização de soluções na fase de projeto do software
* Estabelecer um vocabulário comum, facilitando comunicação, documentação e aprendizado dos sistemas de *software*.

**O Problema**

Digamos que você tenha que fazer um sistema pra um bar especializado em coquetéis, onde existem vários tipos de coquetéis cadastrados. O coquetéis são feitos de uma bebida com vários outros adicionais. As bebidas podem ser cachaça, vodka, tequila, etc.... E os adicionais podem ser limão, leite condensado, açúcar, gelo, etc.... Então pode-se fazer diversas misturas que dependeriam do cliente. Dessa forma, teríamos que criar várias classes para prever o que um possível cliente solicitaria e isso seria inviável. E é aí que entra o decorator...

**Padrão Decorator**

* **O que é?**

**Decorator** é um padrão de projeto de software que permite adicionar um comportamento a um objeto já existente em tempo de execução, ou seja, agrega dinamicamente responsabilidades adicionais a um objeto. Ao contrário da herança que aplica funcionalidades a todos os objetos dela, o padrão decorator permite aplicar funcionalidades apenas a um objeto específico.

* **Características**
* Os decoradores têm o mesmo supertipo que os objetos que eles decoram;
* Você pode usar um ou mais decoradores para englobar um objeto;
* Os objetos podem ser decorados a qualquer momento, então podemos decorar os objetos de maneira dinâmica no tempo de execução com quantos decoradores desejarmos.

**Consequências**

**Impactos positivos**

* Mais flexibilidade do que herança
  + Adição ou remoção de responsabilidades em tempo de execução
  + Adição da mesma propriedade mais de uma vez
* Evita o excesso de funcionalidades nas classes

**Impactos negativos**

* Decorator e seu componente não são idênticos
  + Comparações tornam-se mais complexas
* Resulta em um design que tem vários pequenos objetos, todos parecidos

**Microarquitetura**

Temos uma interface Component que é implementada pela classe concreta ConcreteComponent. As classes Decorator implementam a mesma interface abstrata que o componente que irão decorar. E as classes ConcreteDecorator possuem uma variável de instância para a classe que será decorada e elas podem adicionar novos comportamentos também.

**Padrões Relacionados**

* **Adapter:** Decorator muda comportamento; Adapter muda interface.

O **Adapter** é utilizado para 'adaptar' a interface de uma classe. O Adapter permite que classes com interfaces incompatíveis possam interagir.

Ele permite que um objeto cliente utilize serviços de outros objetos com interfaces diferentes por meio de uma interface única.

* **Composite:** Pode ser visto como um composite com um único componente; porém, não tem intenção de agregar objetos.

**Composite** é utilizado para representar um objeto que é constituído pela composição de objetos similares a ele. Neste padrão, o objeto composto *possui* um conjunto de outros objetos que estão na mesma hierarquia de classes a que ele pertence. O padrão *composite* é normalmente utilizado para representar listas recorrentes - ou recursivas - de elementos.

* **Strategy:** Decorator envolve o objeto; Strategy muda o funcionamento interno.

O **Strategy** permite definir novas operações sem alterar as classes dos elementos sobre os quais opera. Definir uma família de algoritmos e encapsular cada algoritmo como uma classe, permitindo assim que elas possam ser trocados entre si