## *****Design Patterns Logica*****

***#1 -> Strategy***

* Alterar o comportamento em tempo de execução. [Exemplo: Modo de Comer]
* O Strategy Pattern permite que o algoritmo varie independentemente de quem o utilizará.
* No strategy, o cliente é responsável pela troca de comportamento.

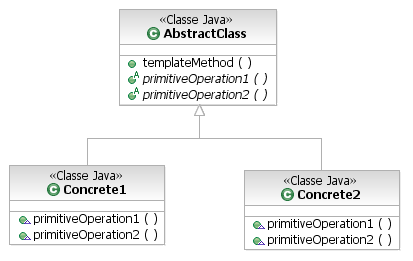
O padrão é aplicado em situações em que muitas classes se relacionam e diferem apenas no modo de atuação, com isso o *Strategy* irá configurar a classe que tenha um dentre muitos comportamentos fornecidos. Também pode ser usado quando há a necessidade da variação de um algoritmo, ou seja, pode-se implementar diferentes códigos que chegam no mesmo objetivo, mas que possuem em determinadas situações mais vantagens do que os demais.



***#2 -> Template Method***

* Altera o comportamento de um método em questão. [Exemplo: Geração de relatórios].
* Realizar uma sequência de passos o qual um dos passos pode ser modificado.
* Abstração do método que será diferente nas outras classes.

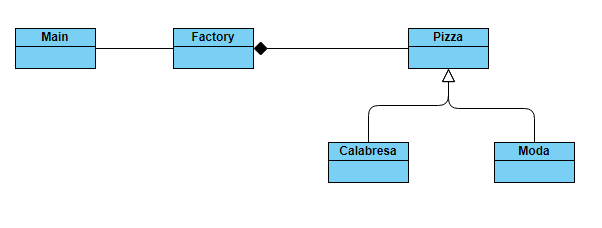
O Padrão Template Method define o esqueleto de um algoritmo dentro de um método, transferindo alguns de seus passos para as subclasses. O Template Method permite que as subclasses redefinam certos passos de um algoritmo sem alterar a estrutura do próprio algoritmo



***#3 -> Factory Method***

Delegar a responsabilidade de instanciar objetos à uma fábrica, diminuindo o acoplamento. [Exemplo: Pizza].

* Quando a classe não antecipa a classe do objeto que quer criar.
* Uma classe quer suas subclasses para especificar os objetos que cria.
* Quando você não quer que o usuário tenha que saber de cada subclasse.
* Encapsular a criação de objetos.

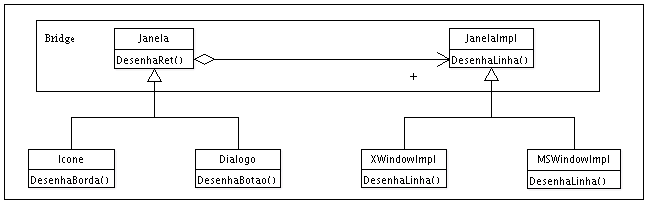


***#4 -> Bridge***

Permitir alteração e variação de dois lados independentemente. [Exemplo: Relatório e modo de exportação]

Bridge tem a intenção de separar a abstração da implementação, ou seja, será utilizado quando temos uma interface que varie muito de implementação.

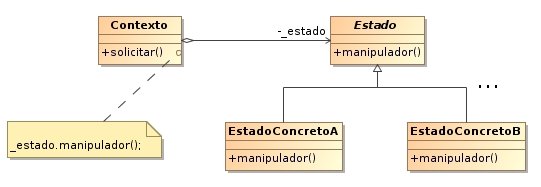
Strategy + Template = Bridge



***#5 -> State***

* Altera o estado de um objeto. [Exemplo: Personagem com pouca vida]
* Semelhante ao strategy permite a troca de "estado" em tempo de execução.
* No strategy, o cliente é responsável pela troca de comportamento.
* No State, a troca pode ser realizada por terceiros (não necessariamente é feita pelo cliente).

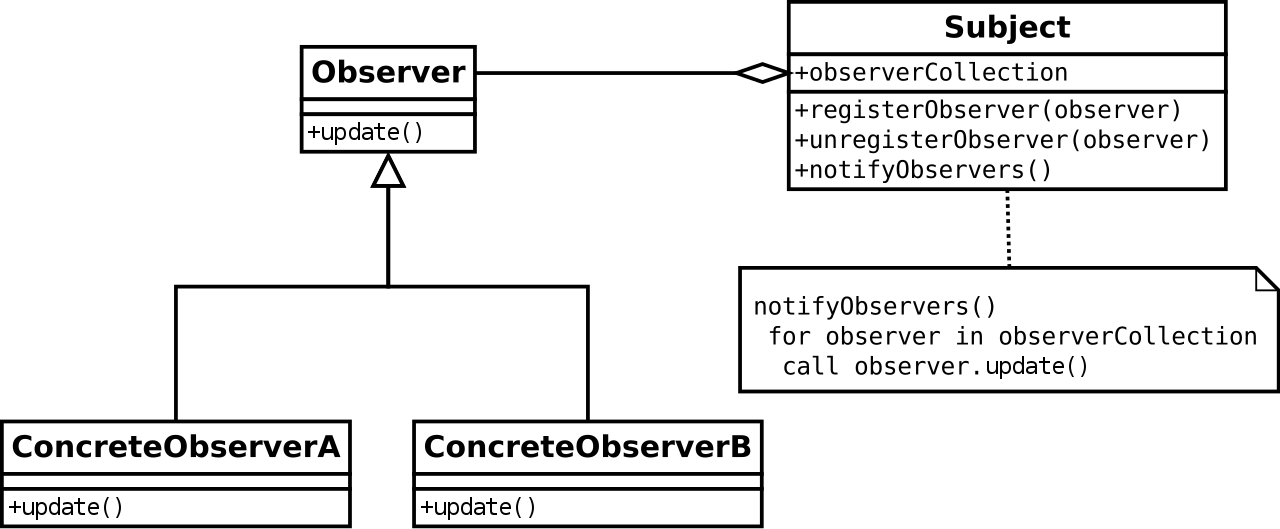
Está recomendado quando um determinado objeto tem estados e responsabilidades diferentes, dependendo de qual estado você está em determinado momento. Também pode ser usada para simplificar os casos em que há código complicado e extenso de decisão que depende do estado do objeto.



***#6 -> Observer***

Notifica objetos observadores sobre alterações no objeto observado. [Exemplo: Termômetro]

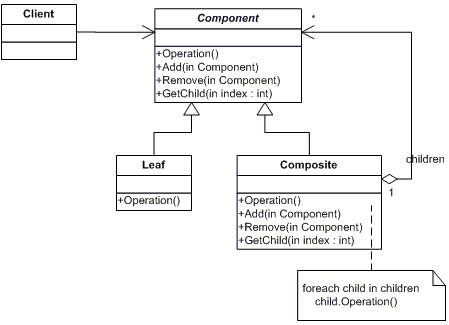
Uma classe que está sendo observada fica responsável por notificar classes que a estão observando quando seu estado mudar.



***#7 -> Composite***

Trata objetos simples e compostos como iguais. [Exemplo: Produto 🡪 Produto Simples, Produto Composto]

A intenção do padrão **Composite** é compor objetos em estruturas de árvore para representar hierarquia partes-todo.



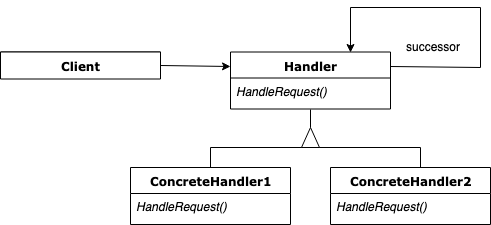
**#8 -> Chain of Responsibility**

Sequência de passos que pode ser alterada em tempo de execução.

Em um sistema orientado a objetos esses interagem entre si através de mensagens, e o sistema necessita de determinar qual o objeto que irá tratar a requisição. O padrão de projeto Chain of Responsibility permite determinar quem será o objeto que irá tratar a requisição durante a execução. Cada objeto pode tratar ou passar a mensagem para o próximo na cascata.

Exemplo: Em um escritório, por exemplo, onde se tem 4 linhas telefônicas, a primeira linha é o primeiro objeto, a segunda linha é o segundo, e assim sucessivamente até a gravação automática que é o quinto objeto. Se a primeira linha estiver disponível ela irá tratar a ligação, se não ela passa a tarefa para o próximo objeto, que é a segunda linha. Se essa estiver ocupada ela passa a tarefa para a próxima e assim sucessivamente até que um objeto possa tratar a tarefa.

Nesse caso, se todas as linhas estiverem ocupadas o último objeto, que é a gravação automática, tratará da tarefa.



***#9 -> Proxy/Decorator***

Filtrar requisições. [Exemplo: Spotify]

