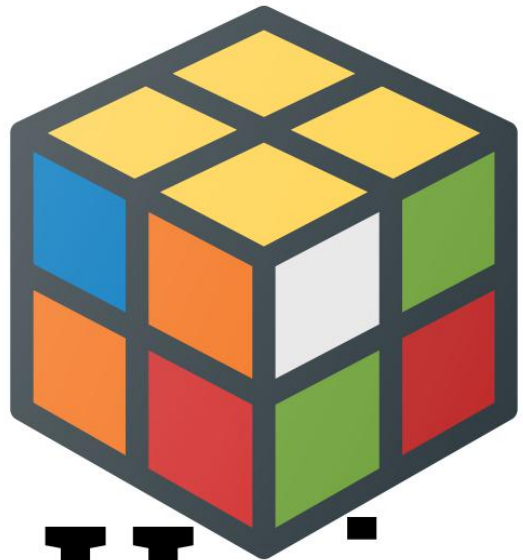


Geek University

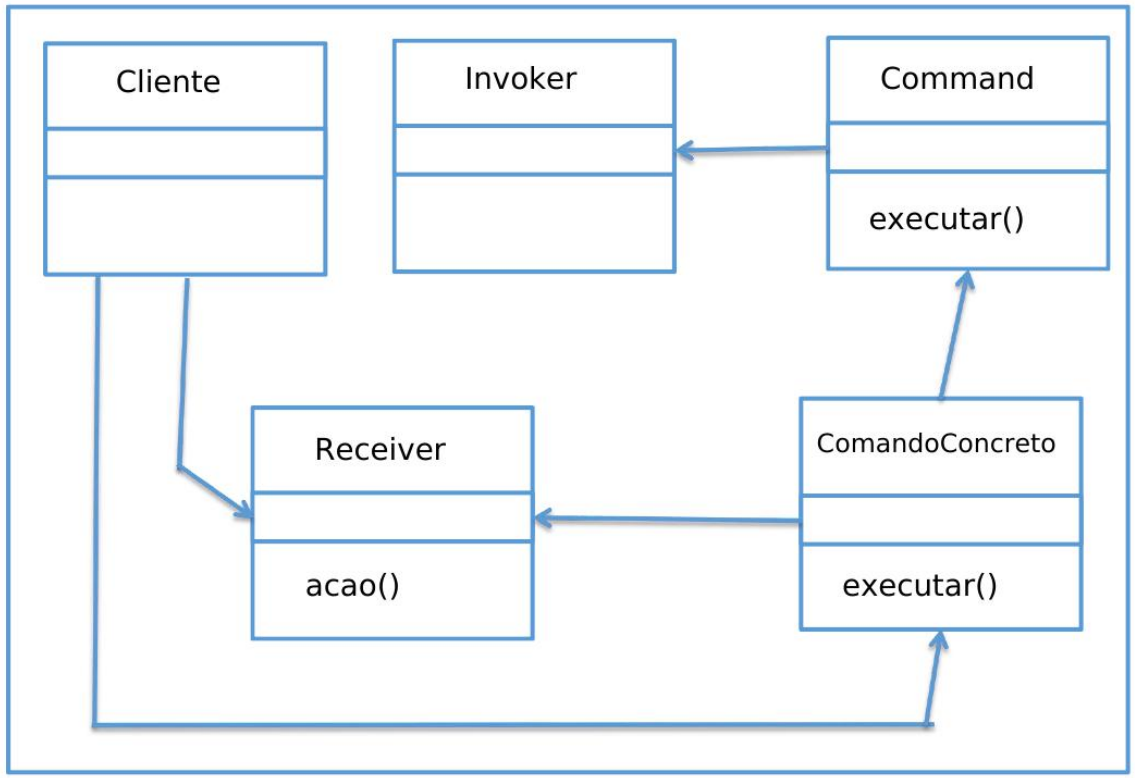
Evolua seu lado geek!

www.geekuniversity.com.br

Considerações no Design de Software

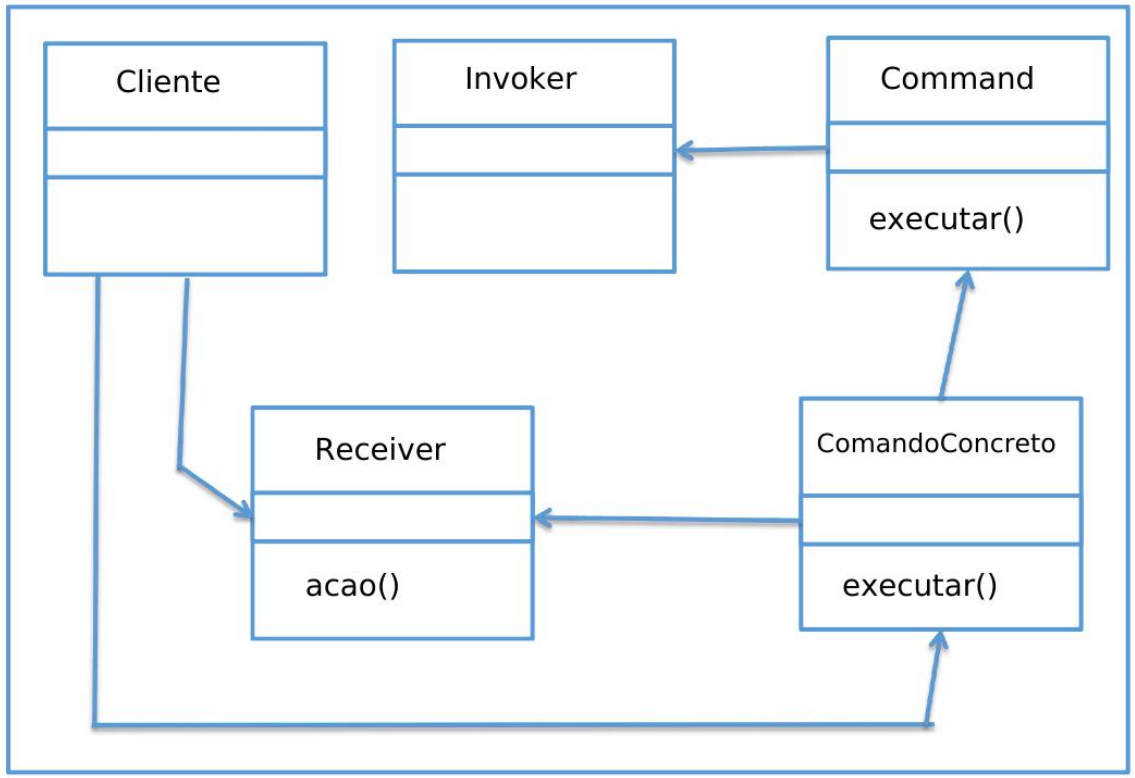
Considerações no Design de Software

Com base no diagrama UML estudado em aulas passadas aprendemos que o padrão Command tem cinco participantes principais: **Cliente**, **Command**, **ComandoConcreto**, **Invoker** e **Receiver**



Considerações no Design de Software

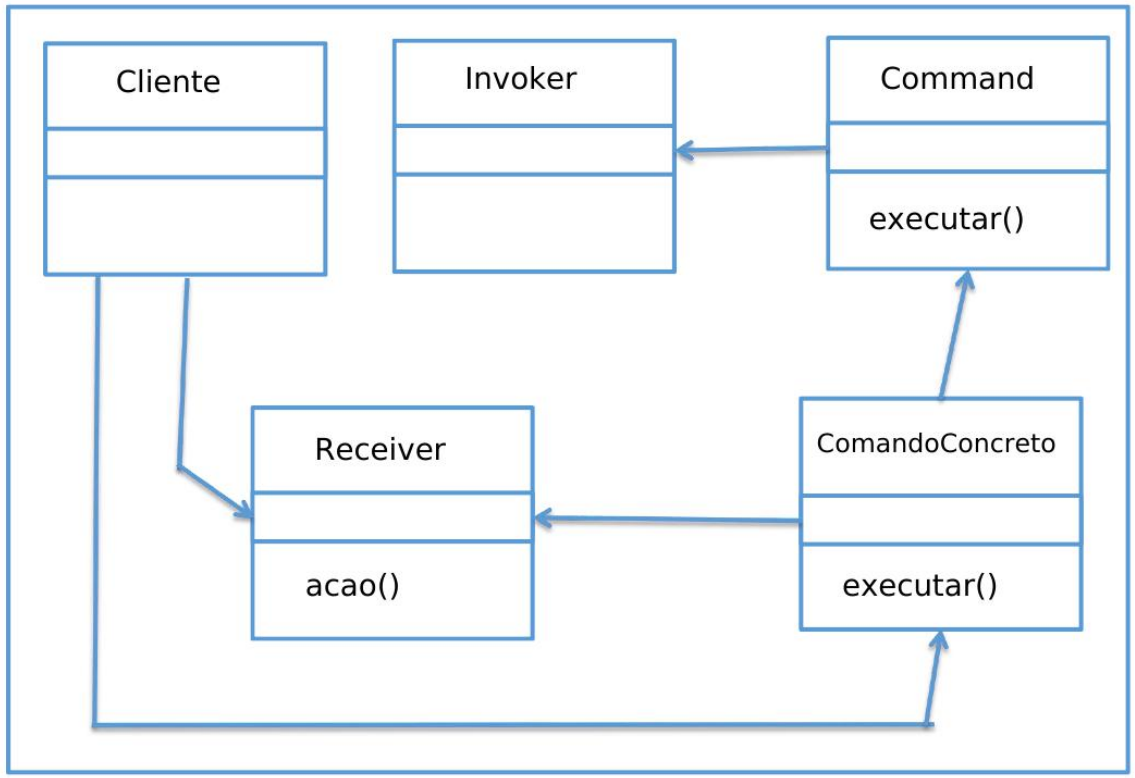
Na nossa última implementação o **Command** do diagrama é representado pela classe/interface **Ordem**.




```python
from abc import ABCMeta, abstractmethod

class Ordem(metaclass=ABCMeta):

    @abstractmethod
    def executar(self):
        pass
```

Considerações no Design de Software

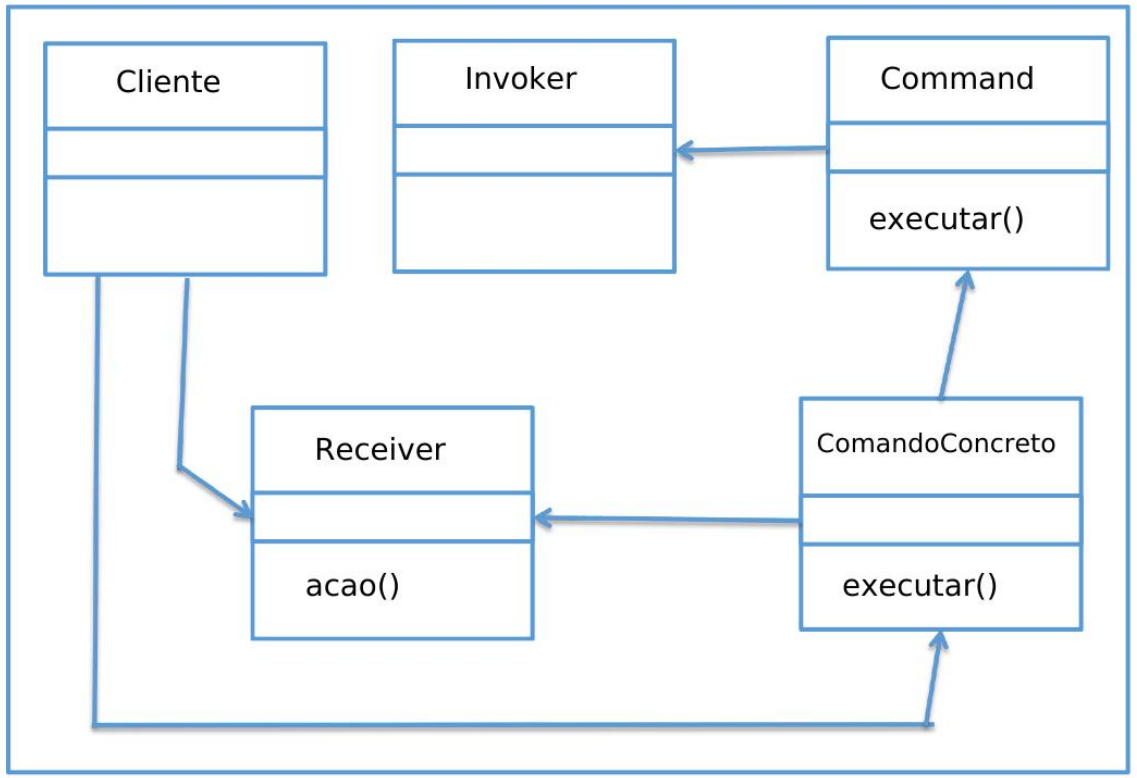
O **Receiver** é representado pela classe **Acao**. Note que apesar do diagrama apresentar apenas um método que executa alguma ação, na nossa implemtação temos dois métodos com ações diferentes.




```python
class Acao:

    def comprar(self):
        print('Você irá comprar ações')

    def vender(self):
        print('Você irá vender ações')
```

Considerações no Design de Software

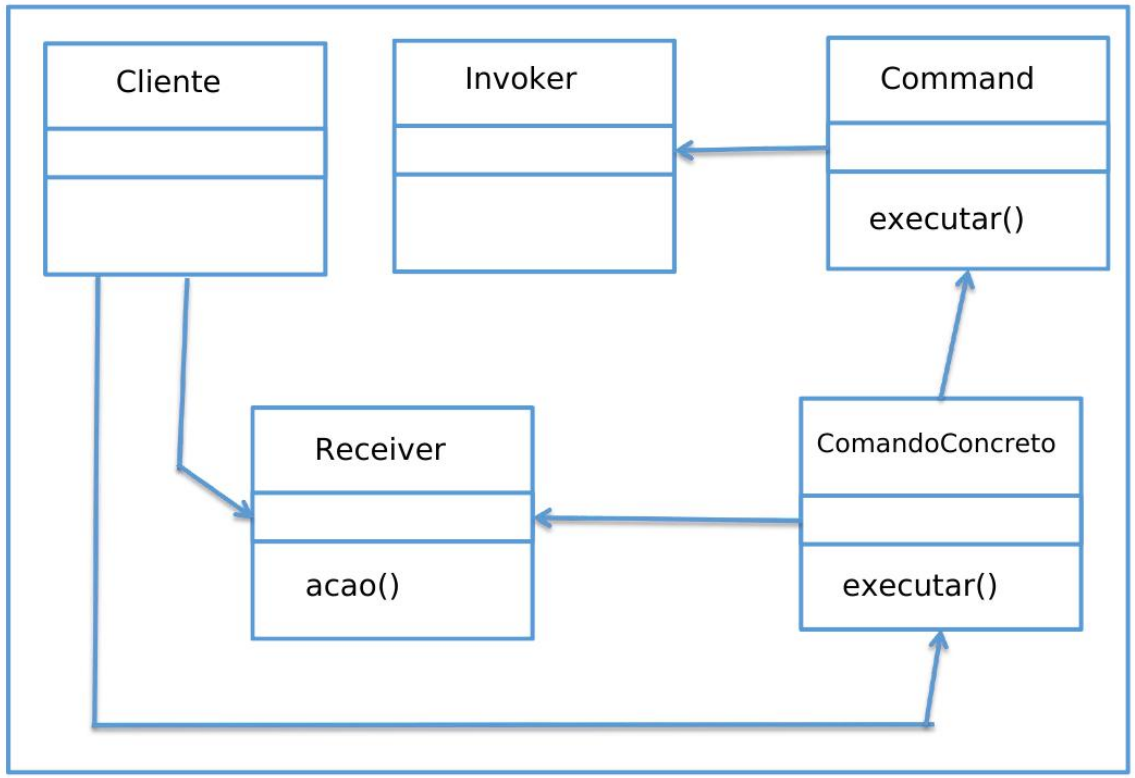
O **Invoker** é representado pela classe **Agente**. Este nada mais é do que um intermediário entre o cliente e a corretora. É este que adiciona as ordens em uma fila de execução e as executa.




```python
class Agente:

    def __init__(self):
        self.__fila_ordens = []

    def adicionar_ordem_na_fila(self, ordem):
        self.__fila_ordens.append(ordem)
        ordem.executar()
```

Considerações no Design de Software

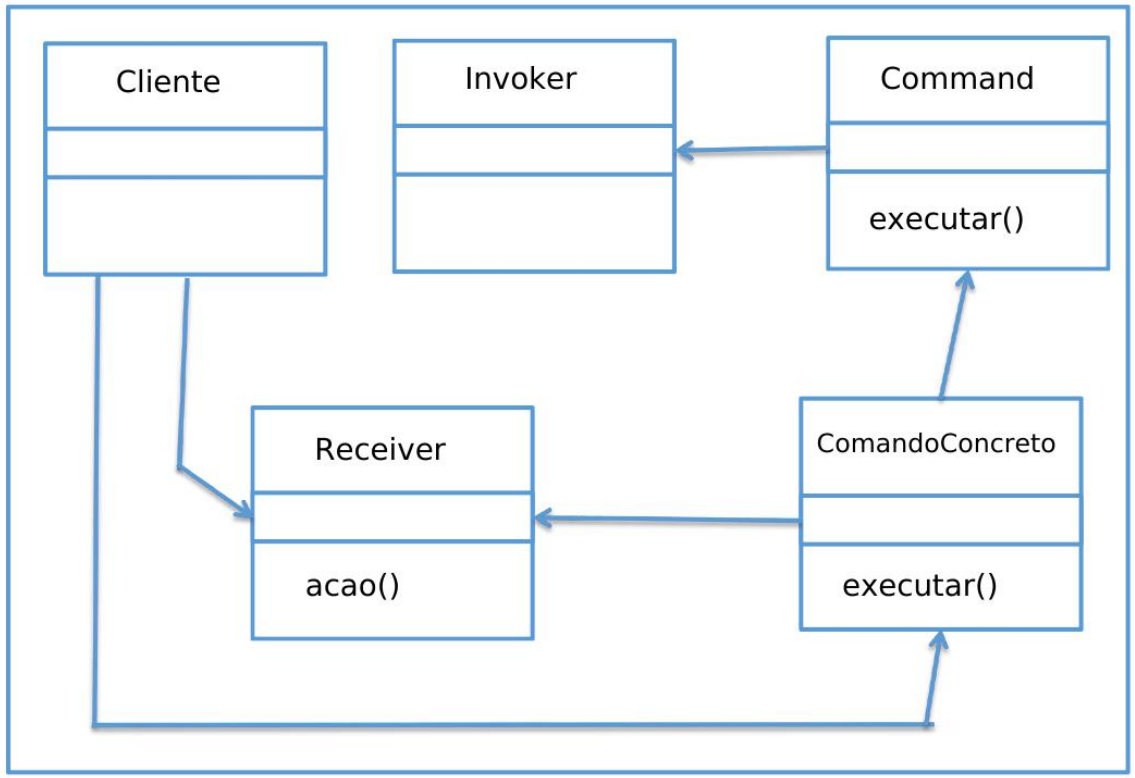
O **ComandoConcreto** é representado pelas classes **OrdemCompra** e **OrdemVenda**. Podemos ter quandos “Comandos Concretos” forem necessários.




```python
class OrdemCompra(Ordem):

    def __init__(self, acao):
        self.acao = acao

    def executar(self):
        self.acao.comprar()


class OrdemVenda(Ordem):

    def __init__(self, acao):
        self.acao = acao

    def executar(self):
        self.acao.vender()
```

Considerações no Design de Software

O **Cliente** cria as ordens de compra e venda da ação e informa ao agente que adiciona na fila de execução.




```python
if __name__ == '__main__':
    # Cliente
    acao = Acao()
    ordem_compra = OrdemCompra(acao)
    ordem_venda = OrdemVenda(acao)

    # Chamador
    agente = Agente()
    agente.adicionar_ordem_na_fila(ordem_compra)
    agente.adicionar_ordem_na_fila(ordem_venda)
```

Considerações no Design de Software

Podemos perceber que apesar do padrão **Command** ter um formato, ele é flexível na implementação.

Isso não ocorre somente com o padrão **Command** mas com praticamente qualquer outro padrão.

Por isso da importância sempre em entender o conceito de cada padrão para que você, a se deparar com algum problema a ser resolvido, possa fazer uso do melhor padrão, caso seja necessário um padrão de software para resolver o problema.

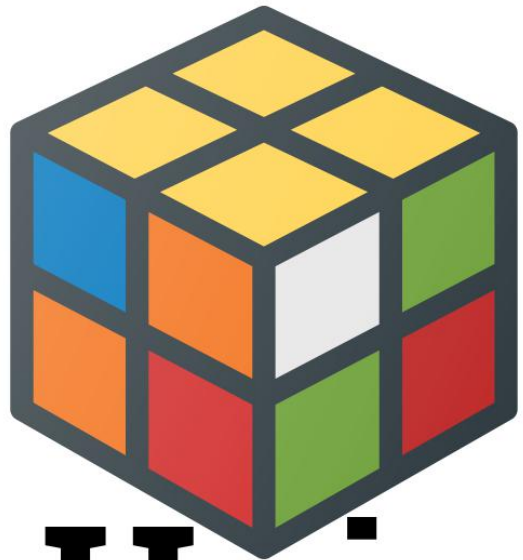
Considerações no Design de Software

Podemos perceber que apesar do padrão **Command** ter um formato, ele é flexível na implementação.

Isso não ocorre somente com o padrão **Command** mas com praticamente qualquer outro padrão.

Por isso da importância sempre em entender o conceito de cada padrão para que você, a se deparar com algum problema a ser resolvido, possa fazer uso do melhor padrão, caso seja necessário um padrão de software para resolver o problema.

Na próxima aula iremos estudar sobre vantagens e desvantagens do padrão Command



Geek University

Evolua seu lado geek!

www.geekuniversity.com.br