Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 14 Padrões de Comportamento - Command

Intenção

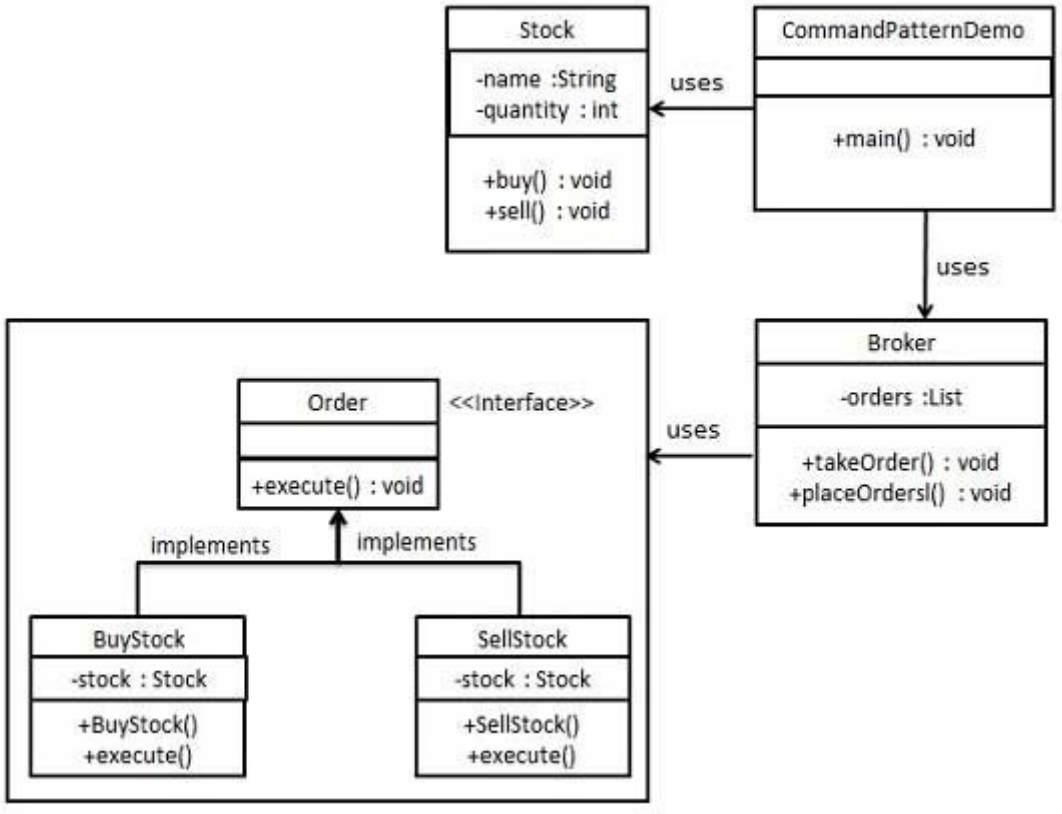
Encapsular uma requisição como um objeto, permitindo parametrização, enfileiramento, suporte a histórico, etc. Também conhecido como: Action, Transaction

Usar este padrão quando…

- Quiser parametrizar ações genéricas;
- Quiser enfileirar e executar comandos de forma assíncrona, em outro momento;
- Quiser permitir o undo de operações, dando suporte a históricos;
- Quiser fazer log dos comandos para refazê-los em caso de falha de sistema;
- Quiser estruturar um sistema em torno de operações genéricas, como transações.

Vantagens e desvantagens

- Desacoplamento:
  - Objeto que evoca a operação e o que executa são desacoplados.
- Extensibilidade:
  - Comandos são objetos, passíveis de extensão, composição, etc.;
  - Pode ser usado junto com Composite para formar comandos complexos;
  - É possível definir novos comandos sem alterar nada existente.



Passo 1
Crie uma interface de comando.

Order.java

```
public interface Order {
   void execute();
}
```

Passo 2
Crie uma classe de solicitação.

Stock.java

```
public class Stock {

   private String name = "ABC";
   private int quantity = 10;

   public void buy(){
      System.out.println("Stock [ Name: "+name+",
         Quantity: " + quantity +" ] bought");
   }
   public void sell(){
      System.out.println("Stock [ Name: "+name+",
         Quantity: " + quantity +" ] sold");
   }
}
```

Passo 3
Crie classes concretas implementando a interface Order.

BuyStock.java

```
public class BuyStock implements Order {
   private Stock abcStock;

   public BuyStock(Stock abcStock){
      this.abcStock = abcStock;
   }

   public void execute() {
      abcStock.buy();
   }
}
```

SellStock.java

```
public class SellStock implements Order {
   private Stock abcStock;

   public SellStock(Stock abcStock){
      this.abcStock = abcStock;
   }

   public void execute() {
      abcStock.sell();
   }
}
```

Passo 4
Criar classe de chamador de comando.

Broker.java

```
import java.util.ArrayList;
import java.util.List;

   public class Broker {
   private List<Order> orderList = new ArrayList<Order>();

   public void takeOrder(Order order){
      orderList.add(order);
   }

   public void placeOrders(){

      for (Order order : orderList) {
         order.execute();
      }
      orderList.clear();
   }
}
```

Passo 5
Use a classe Broker para obter e executar comandos.

CommandPatternDemo.java

```
public class CommandPatternDemo {
   public static void main(String[] args) {
      Stock abcStock = new Stock();

      BuyStock buyStockOrder = new BuyStock(abcStock);
      SellStock sellStockOrder = new SellStock(abcStock);

      Broker broker = new Broker();
      broker.takeOrder(buyStockOrder);
      broker.takeOrder(sellStockOrder);

      broker.placeOrders();
   }
}
```

Passo 6

Teste sua implementação!