Design Patterns - Command Pattern

Command pattern is a data driven design pattern and falls under behavioral pattern category. A request is wrapped under an object as command and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.

命令模式是一种数据驱动的设计模式，属于行为型模式这一类。命令模式会将一个请求包装成一个对象并以命令的方式传递给调用者对象。调用者对象会寻找合适的并且能够处理该命令的对象，然后把该命令传递给相应的对象处理。

Implementation

实现

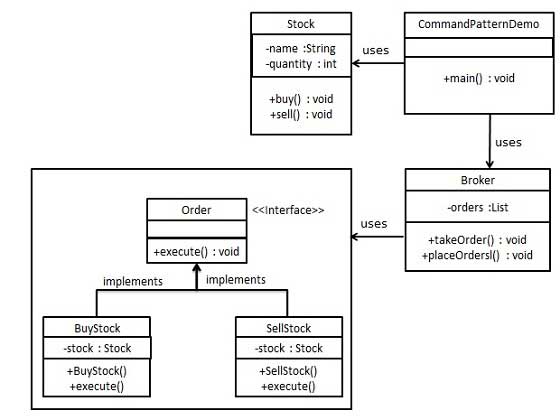
We have created an interface *Order* which is acting as a command.

We have created a *Stock* class which acts as a request. We have concrete command classes *BuyStock* and*SellStock* implementing *Order* interface which will do actual command processing. A class *Broker* is created which acts as an invoker object. It can take and place orders.

*Broker* object uses command pattern to identify which object will execute which command based on the type of command.*CommandPatternDemo*, our demo class, will use *Broker* class to demonstrate command pattern.

我们创建了一个Order接口，该接口代表一组命令。紧接着创建一个Stock类代表请求。创建具体的命令类 BuyStock和SellStock 实现Order接口，它们将会作为具体的命令被处理。 Broker 代表调用者，它能获得并且发出命令。

Broker对象将通过命令模式来识别哪种对象该执行哪种命令。CommandPatternDemo，我们的demo类，将使用Broker类来演示命令模式。



Step 1

第一步

Create a command interface.

创建命令接口。

*Order.java*

public interface Order {

void execute();

}

Step 2

第二步

Create a request class.

创建一个请求类

*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");

}

}

Step 3

第三步

Create concrete classes implementing the *Order* interface.

创建具体类实现Order接口

*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();

}

}

Step 4

第四步

Create command invoker class.

创建命令调用者类

*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();

}

}

Step 5

第五步

Use the Broker class to take and execute commands.

使用Broker类获得并且执行命令

*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();

}

}

Step 6

第六步

Verify the output.

校验输出。

Stock [ Name: ABC, Quantity: 10 ] bought

Stock [ Name: ABC, Quantity: 10 ] sold