Command

```
// Command (nell'esempio della ristorazione corrrisponde all'ordine da cucinare)
public interface Order {
	public void execute();
}
// Invoker (nell'esempio della ristorazione corrisponde al cameriere, che prende l'ordine e
porta il piatto pronto)
public class Broker {
	private ArrayList<Order> orders = new ArrayList<>();
	public void takeOrder(Order order) {
		orders.add(order);
	}
	public void placeOrders() {
		for(Order order : orders) {
			order.execute();
		}
		orders.clear();
	}
}
// Request (nell'esempio della ristorazione corrisponde al singolo ordine)
public class Stock {
	private String name;
	private int qta;
	public Stock(String name, int qta) {
		this.name = name;
		this.qta = qta;
	}
	public void buy() {
		System.out.println("[BOUGHT STOCK]: " + name + ", " + qta);
	}
	public void sell() {
		System.out.println("[SOLD STOCK]: " + name + ", " + qta);
	}
}
// Concrete Command (actual command processing), (corrisponde allo chef di quello specifico
piatto)
public class BuyStock implements Order {
	private Stock stock;
	public BuyStock(Stock stock) {
		this.stock = stock;
	}
	@Override
	public void execute() {
		stock.buy();
	}
}
//Concrete Command (actual command processing)
public class SellStock implements Order {
	private Stock stock;
	public SellStock(Stock stock) {
		this.stock = stock;
	}
	@Override
	public void execute() {
		stock.sell();
	}
}
```

```
public class Main {
	public static void main(String[] args) {
		Stock stock = new Stock("Skate", 10);
		BuyStock buy = new BuyStock(stock);
		SellStock sell = new SellStock(stock);
		Broker broker = new Broker();
		broker.takeOrder(buy);
		broker.takeOrder(sell);
		broker.placeOrders();
	}
}
```