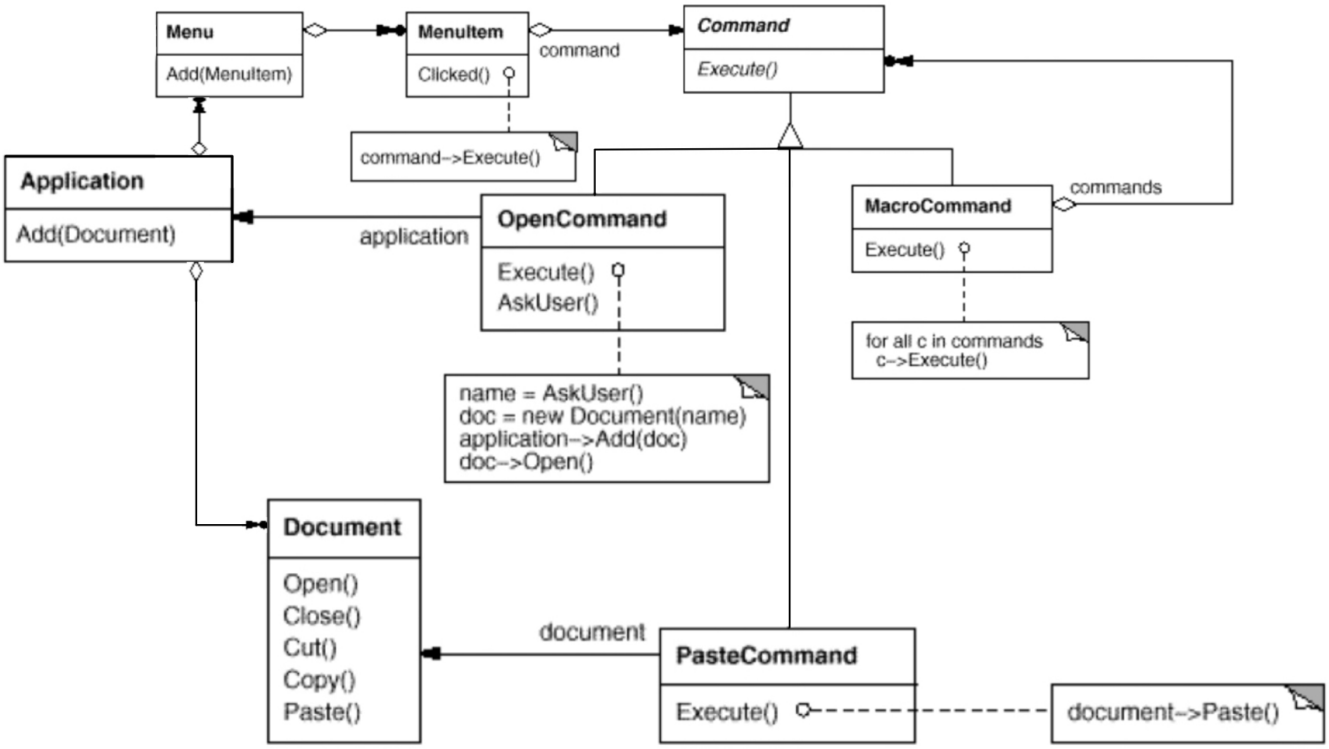
Command

↓ INTENT ↓

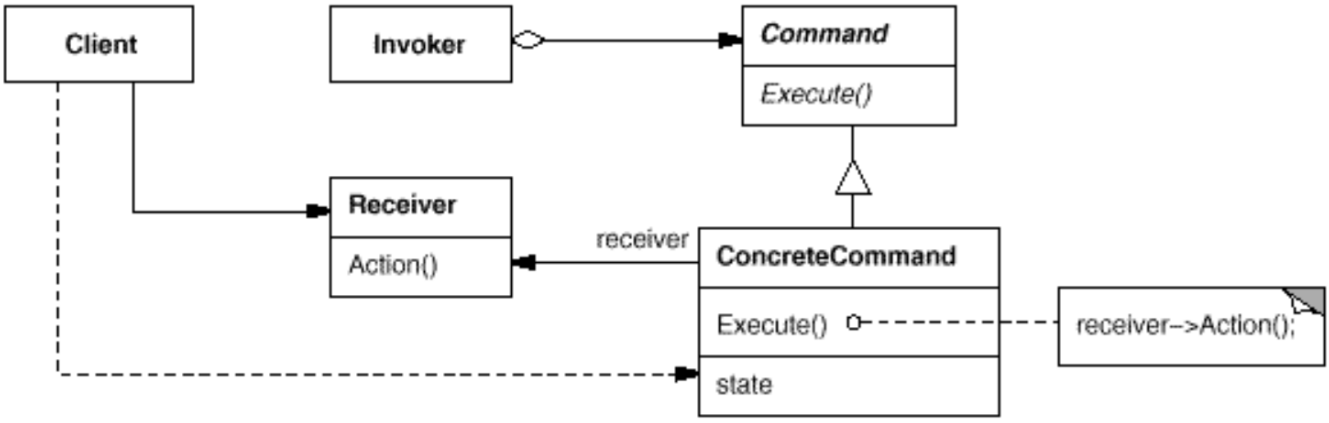
```
Incapsula una richiesta come oggetto, consentendo in tal modo di
parametrizzare i client con richieste diverse, richieste di coda o di
registro e operazioni non supportabili.

(nel progetto da usare quando si utilizzano delle query)
```

↓ MOTIVATION ↓

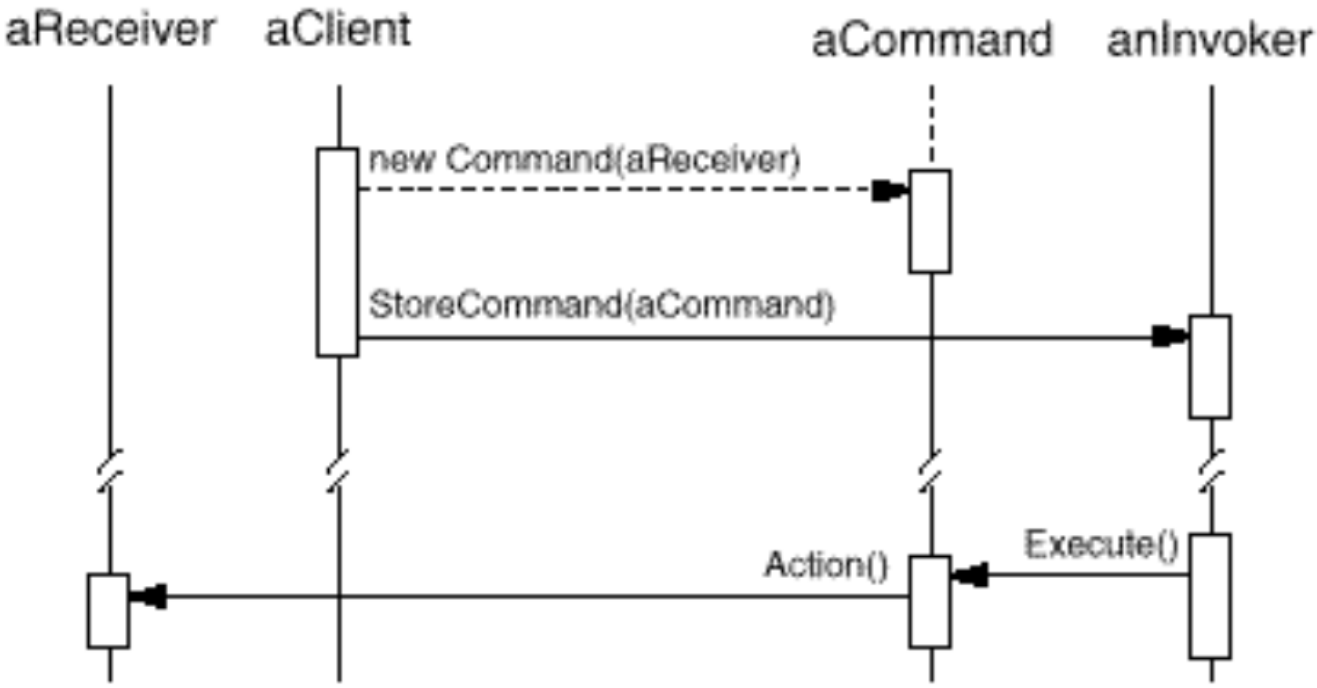


↓ STRUCTURE ↓

hh ↓



↓ COLLABORATION ↓



↓ EXAMPLE ↓

Order ↓

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

BuyStock ↓

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

```
    public BuyStock(Stock stock) {
        this.stock = stock;
    }

    @Override
    public void execute() {
        stock.buy();
    }

}
```

SellStock ↓

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

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

    @Override
    public void execute() {
        stock.sell();
    }

}
```

Stock ↓

```
public class Stock {
    private String name;
    private int quantity;

    public Stock(String name, int quantity) {
        this.name = name;
        this.quantity = quantity;
    }

    public void buy(){
        System.out.println("comprati " + name + " che contine " + quantity
+ " prodotti");
    }

    public void sell(){
        System.out.println("venduti " + name + " che contine " + quantity
+ " prodotti");
    }
}
```

Broker ↓

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

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

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

Main ↓

```
public static void main(String[] args) {
    Stock stock = new Stock("prodotti", 10);

    BuyStock buyStock  = new BuyStock(stock);
    SellStock sellStock = new SellStock(stock);

    Broker broker = new Broker();
    broker.takeOrder(buyStock);
    broker.takeOrder(sellStock);

    broker.placeOrder();
}
```