02

Diferentes ações com Command

Imagine um sistema que controla pedidos de uma empresa. O pedido, após ser feito pelo site, passa por diversas etapas de um workflow interno. Por exemplo, um pedido precisa ser processado e, depois de pago, os itens devem ser separados, e depois entregues ao cliente.

Vamos representar essa classe em Java:

```
public class Pedido {

    private String cliente;
    private double valor;
    private Status status;
    private Calendar dataFinalizacao;


    public Pedido(String cliente, double valor) {
        this.cliente = cliente;
        this.valor = valor;
    }

    public void paga() {
        status = Status.PAGO;
    }

    public void finaliza() {
        dataFinalizacao = Calendar.getInstance();
        status = Status.ENTREGUE;
    }
}

public enum Status {

    NOVO,
    PROCESSANDO,
    PAGO,
    ITEM_SEPARADO,
    ENTREGUE;
}
```

Agora imagine que essa loja receba muitas requisições. Para tratá-las, é necessário fazer uso de algum tipo de fila, que executa diferentes comandos para diferentes pedidos.

Vamos implementar esse leitor da fila:

```
public class FilaDeTrabalho {

    private List<Pedido> pedidos;
}
```

Mas o problema é que não adianta guardar somente o pedido, mas sim a ação que precisamos executar em cima dele. Algo como:

```
public class FilaDeTrabalho {

    private List<Comando> comandos;
}
```

Vamos então criar a interface Comando, que representará um Comando que deve ser executado:

```
public interface Comando {
    void executa();
}
```

Agora vamos criar os comandos. Um deles finalizará o pedido, o outro deles marca o pedido como pago:

```
public class ConcluiPedido implements Comando {

    private Pedido pedido;

    public ConcluiPedido(Pedido pedido) {
        this.pedido = pedido;
    }

    @Override
    public void executa() {
        pedido.finaliza();
    }

}

public class PagaPedido implements Comando {

    private Pedido pedido;

    public PagaPedido(Pedido pedido){
        this.pedido = pedido;
    }
    @Override
    public void executa() {
        pedido.paga();
    }

}
```

Repare que cada comando recebe um Pedido, e já sabe exatamente qual método invocar.

Dessa forma, temos uma "fila de comandos" a ser executada, e podemos executar da maneira que acharmos melhor. Vamos melhorar um pouco a classe que cuida da fila, e dar métodos para adicionar e processar todos:

```
public class FilaDeTrabalho {
```

```
    private List<Comando> comandos;

    public FilaDeTrabalho() {
        comandos = new ArrayList<Comando>();
    }

    public void adiciona(Comando comando) {
        comandos.add(comando);
    }

    public void processa() {
        for(Comando comando : comandos) {
            comando.executa();
        }
    }
}
```

Agora, um simples programa para testar seu uso:

```
public class Programa {

    public static void main(String[] args) {
        Pedido pedido1 = new Pedido("Mauricio", 150.0);
        Pedido pedido2 = new Pedido("Marcelo", 250.0);

        FilaDeTrabalho fila = new FilaDeTrabalho();

        fila.adiciona(new PagaPedido(pedido1));
        fila.adiciona(new PagaPedido(pedido2));
        fila.adiciona(new ConcluiPedido(pedido1));

        fila.processa();
    }
}
```

Pronto. Agora temos uma fila, que executa comandos em cima dos nossos pedidos. E executá-los ficou fácil. Criar novos comandos também é fácil.

O nome desse padrão de projeto, que facilita a criação de comandos, chama-se **Command**. Usamos ele quando temos que separar os comandos que serão executados do objeto que ele pertence. Um bom exemplo disso é o uso de filas de trabalho.