COMMAND PATTERN

Problema inicial

"Crear una oficina que acepte pedidos, los registre y procese según requiera el peso del paquete"
Post data: "Muy fácil. Siempre será así. No se complicará"

```
public interface Pedido {

  int peso();

}
```

```
public class Oficina {


  public void recibe (Pedido pedido) {
    System.out.println("registrando pedido");
    if (pedido.peso() > 20 ) {
      // tratamiento paquete grande
    } else {
      // tratamiento paquete normal
    }
    System.out.println("registrar si ha ido bien");
  }

}
```

Problema inicial

"umm ... te dije que los paquetes pueden ser internacionales y van a parte?"

```
public interface Pedido {

  int peso();

  boolean internacional();

}
```

¿Que pattern podemos usar?

¿Un *chain of responsability* ?

```
public interface ProcesadorPedido {

  boolean acepta (Pedido pedido);

  boolean trata (Pedido pedido);

}
```

Mucho mejor. ¿Estamos ya a salvo?

“Por cierto, tambien hay un tipo de paquetes especiales con mercancias peligrosas. Éste tipo de paquetes (y solo este) lleva instrucciones de manejo específicas.

Y ya. Nada más.

Excepto que algunos paquetes serán urgentes ... pero esto seguro que ya lo hacias, ¿no?

Esto ... hemos hablado de los paquetes que se tratan en grupo, ¿no?

— *El jefe*”

Veamos ... pedidos peligrosos ...

```
public interface PedidoPeligroso extends Pedido {

     String instrucciones();

}
```

Podemos tratarlo aún con el chain of responsability ... habrá que hacer un instanceof (feo) pero se puede tratar

Siguiente ... pedidos urgentes ...

```
public interface Pedido {

  int peso();

  boolean urgente();

}
```

Parece que la interface Pedido no es muy estable
(¿que principio viola?)
.... peligro peligro

Empieza a ser delicado el orden en el chain of responsability

Siguiente ... ¡ un pedido múltiples paquetes !

```
public interface ProcesadorPedido {

  boolean acepta (Pedido pedido);

  boolean trata (Pedido pedido);

}
```

No podemos anticipar un proceso común para tratar todos los tipos de pedidos

```
public class Oficina {


  public void recibe (...) {
    System.out.println("registrando pedido");

      // tratamiento específico


    System.out.println("registrar si ha ido bien");
  }

}
```

La parte flexible (móbil), no puede ser el tipo de paquete, sino como se trata el pedido

“

Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

Aka: Transaction

— *GoF*

”

Command pattern al rescate

```
public interface TratamientoPedido {

  boolean tratar();
}
```

⟵ Representa una operación, no un "objeto"

```
public class Oficina {

  public void recibe (TratamientoPedido pedido) {
    System.out.println("registrando pedido");

    boolean ok = pedido.tratar();

    System.out.println("registrar si ha ido bien");
  }

}
```

```java
package org.formacion.command;

public class TratamientoPedidoPeligroso implements TratamientoPedido {

  private PedidoPeligroso pedido;

  public TratamientoPedidoPeligroso(PedidoPeligroso pedido) {
    this.pedido = pedido;
  }


  public boolean tratar() {
    // tratar el pedido segun las instrucciones
    return true; // si es false ... :-(
  }
}
```

```java
package org.formacion.command;

import java.util.List;

public class TratamientoPedidoMultiple implements TratamientoPedido {

  private List<Pedido> pedidos;

  public TratamientoPedidoMultiple(List<Pedido> pedidos) {
    this.pedidos = pedidos;
  }



  public boolean tratar() {
    // iterar sobre la lista de pedidos y tratar cada uno
    return true;
  }
}
```

COMMAND PATTERN