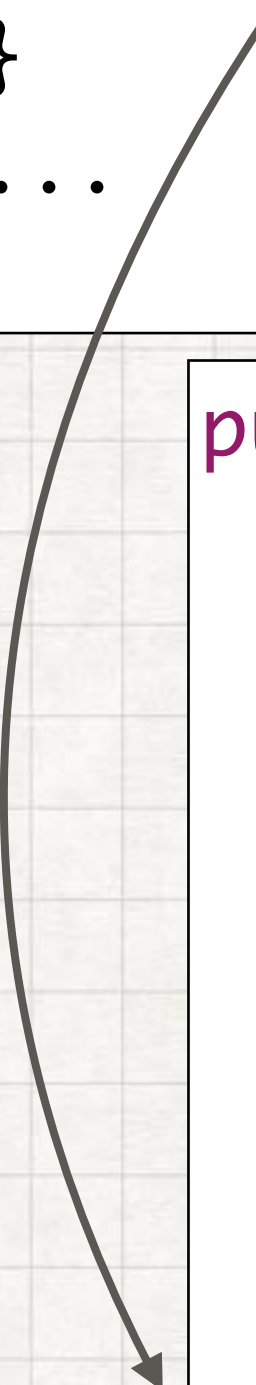


MEMENTO PATTERN


```
public class CestaCompra {  
  
    private Map<Producto, LineaCompra> lista = new LinkedHashMap<>();  
  
    public void add (Producto producto) {  
        ...  
    }  
    ...  
}
```



```
public class LineaCompra {  
  
    private Producto producto;  
    private int cantidad;  
    private int precio;  
  
    ...  
  
    public void incrementa() {  
        cantidad++;  
        precio += producto.getPrecio();  
    }  
}
```

```
public class Producto {  
  
    private String nombre;  
    private int precio;  
    ...  
}
```



```
CestaCompra cesta = new CestaCompra();  
  
Producto leche = new Producto("leche", 100);  
Producto galletas = new Producto("galletas", 120);  
  
cesta.add(leche);  
cesta.add(galletas);  
cesta.add(leche);  
cesta.add(leche);  
  
System.out.println(cesta);
```

← API de CestaCompra: add(producto) y toString() ... o similar



```
leche x3 : 300  
galletas x1 : 120
```


“

Debe ser posible deshacer el último
producto añadido

— *El jefe*

”


```
public class CestaCompra {  
  
    private Map<Producto, LineaCompra> lista = new LinkedHashMap<>();  
    private Producto ultimo;  
  
    public void add (Producto producto) {  
        ...  
        ultimo = producto;  
    }  
  
    public void retrocede() {  
        lista.get(ultimo).decrementa();  
    }  
}
```

*Guarda cual es el último
producto añadido*

```
public class LineaCompra {  
  
    private Producto producto;  
    private int cantidad;  
    private int precio;  
  
    ...  
  
    public void decrementa() {  
        cantidad--;  
        precio -= producto.getPrecio();  
    }  
}
```

*Esquema de funcionamiento: faltaría
tratamiento de llamadas incorrectas, limpieza
de productos con cantidad 0, etc*

“

Algunos productos tienen oferta según
la cantidad de unidades compradas

— *El jefe*

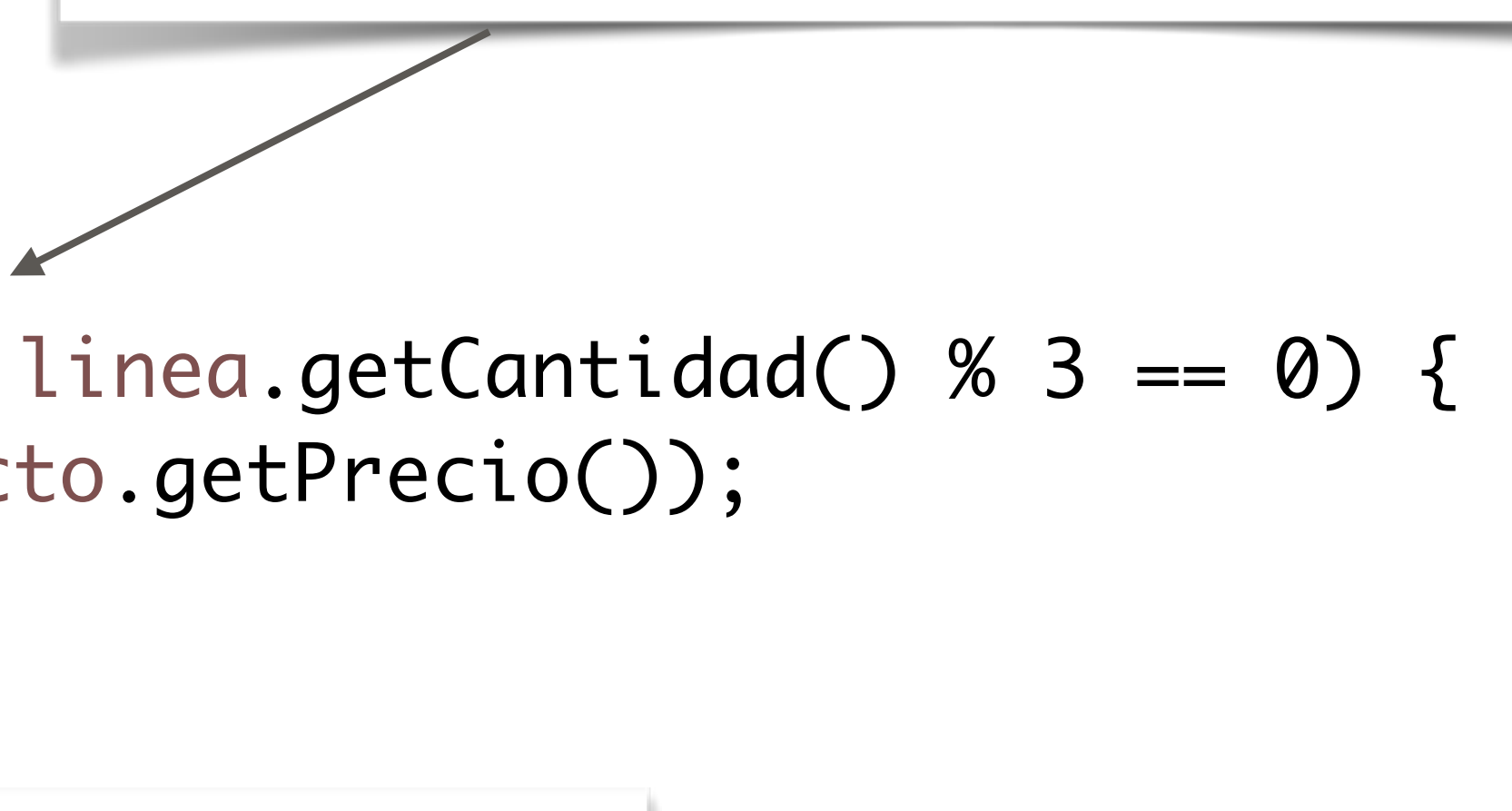
”

Una oferta actúa sobre una línea de compra: modifica los datos si se aplica la oferta

```
public interface Oferta {  
    void aplica (LineaCompra linea);  
}
```

```
public class Oferta3x2Leche implements Oferta {  
    @Override  
    public void aplica(LineaCompra linea) {  
        Producto producto = linea.getProducto();  
        if (producto.getNombre().equals("leche") && linea.getCantidad() % 3 == 0) {  
            linea.setPrecio(linea.getPrecio() - producto.getPrecio());  
        }  
    }  
}
```

El producto es leche y la unidad recién añadida es múltiple de 3 (no debe cobrarse)



Descontamos la última unidad


```
public class CestaCompra {  
  
    private Map<Producto, LineaCompra> lista = new LinkedHashMap<>();  
    private List<Oferta> ofertas = new ArrayList<>();  
  
    private Producto ultimo;  
  
    public void add (Oferta oferta) {  
        ofertas.add(oferta);  
    }  
  
    public void add (Producto producto) {  
        ... añadir producto a la linea  
  
        for (Oferta oferta: ofertas) {  
            oferta.aplica(linea);  
        }  
  
        ultimo = producto;  
    }  
    ..  
}
```

← Añadido a la API: poder incluir ofertas que se aplican

← Finalizado el tratamiento "normal", miramos que ofertas pueden aplicarse


```
CestaCompra cesta = new CestaCompra();

cesta.add(new Oferta3x2Leche());

Producto leche = new Producto("leche", 100);
Producto galletas = new Producto("galletas", 120);

cesta.add(leche);
cesta.add(galletas);
cesta.add(leche);
cesta.add(leche);

System.out.println(cesta);

cesta.retrocede();

System.out.println(cesta);
```

leche x3 : 200
galletas x1 : 120

OK

leche x2 : 100
galletas x1 : 120

ERROR: son 200

```
public void decrementa() {
    cantidad--;
    precio -= producto.getPrecio();
}
```

Problema: no hay forma fácil y general de aplicar
operación "inversa"

“

Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

— *GoF*

”

(...) **capture** and externalize an object's internal state so that the object can be **restored** to this state later.

Simplificación: suponemos que el estado de la cesta incluye la lista de compra pero no las ofertas

```
public class CestaCompra {  
    private Map<Producto, LineaCompra> lista = new LinkedHashMap<>();  
    ....  
    public Map<Producto, LineaCompra> getList() {  
        return lista;  
    }  
    public void setLista(Map<Producto, LineaCompra> lista) {  
        this.lista = lista;  
    }  
}
```

← Obtener estado

← Restaurar estado

Problemas:

1.No funciona

2. ...

*Lo veremos en un
momento ...*

“

Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

— GoF

”


```
public class CestaCompra {  
  
    private Map<Producto, LineaCompra> lista = new LinkedHashMap<>();  
    private List<Oferta> ofertas = new ArrayList<>();  
    ...  
  
    public Memento creaEstado() {  
        return new Memento(lista);  
    }  
  
    public void restaura(Memento memento) {  
        lista = memento.estado;  
    }  
  
    public static class Memento {  
        private final Map<Producto, LineaCompra> estado;  
  
        private Memento(Map<Producto, LineaCompra> estado) {  
            this.estado = estado;  
        }  
    }  
}
```

← Obtener estado

← Restaurar estado

Todo en la clase Memento es privado. Las clases externas solo pueden obtener "mementos" y usarlos para restaurar una cesta.

La estructura interna de CestaCompra sigue siendo privada

... pero habíamos dicho que esto fallaba ...

```
CestaCompra cesta = new CestaCompra();  
  
cesta.add(new Oferta3x2Leche());  
  
Producto leche = new Producto("leche", 100);  
Producto galletas = new Producto("galletas", 120);  
  
cesta.add(leche);  
cesta.add(galletas);  
cesta.add(leche);  
  
Memento anterior = cesta.creaEstado();  
  
cesta.add(leche);  
  
System.out.println(cesta);  
  
cesta.restaura(anterior);  
  
System.out.println(cesta);
```

```
public static class Memento {  
    ...  
    private Memento(... estado) {  
        this.estado = estado;  
    }  
}
```

Problema: el "estado" guardado apunta al mismo Map. El siguiente add ha modificado tanto la lista actual como la del memento

leche x3 : 200
galletas x1 : 120

leche x3 : 200
galletas x1 : 120

ERROR: son x2


```
public static class Memento {  
  
    private final Map<Producto, LineaCompra> estado;  
  
    private Memento(Map<Producto, LineaCompra> estado) {  
        this.estado = new LinkedHashMap<>(estado.size());  
        for (Map.Entry<Producto, LineaCompra> each: estado.entrySet()) {  
            LineaCompra linea = each.getValue();  
            this.estado.put( each.getKey(),  
                            new LineaCompra( linea.getProducto(),  
                                              linea.getCantidad(),  
                                              linea.getPrecio()));  
        }  
    }  
}
```

Consideramos que compartir
producto no es problema

this.estado no comparte ni Map
ni LineaCompra con el original,
crea un nuevo map y una nueva
linea para cada una existente

Técnica: *Deep copy*

En lugar de copiar la referencia a la estructura anterior o a objetos anteriores, propagamos la copia por todos los objetos, asegurándonos que la copia creada no comparte referencias a partes que no nos interesan

La versión anterior hacía un
Shallow copy


```
CestaCompra cesta = new CestaCompra();  
  
cesta.add(new Oferta3x2Leche());  
  
Producto leche = new Producto("leche", 100);  
Producto galletas = new Producto("galletas", 120);  
  
cesta.add(leche);  
cesta.add(galletas);  
cesta.add(leche);  
  
Memento anterior = cesta.creaEstado();  
  
cesta.add(leche);  
  
System.out.println(cesta);  
  
cesta.restaura(anterior);  
  
System.out.println(cesta);
```

Con el objeto memento no puedo hacer nada, solo obtener una copia que puedo usar para restaurar la cesta de a compra a un estado anterior.

La estructura interna de CestaCompra sigue encapsulada.
Puede evolucionar sin romper el código que usa el memento

```
leche x3 : 200  
galletas x1 : 120
```

```
leche x2 : 200  
galletas x1 : 120
```

OK