

DECORATOR PATTERN

Servimos tres tipos de café: cafe corto, largo y descafeinado. Cada uno a un precio.

```
public interface Cafe {  
    String descripcion();  
    int precio();  
}
```

```
public class CafeCorto implements Cafe {  
    public String descripcion() {  
        return "cafe corto";  
    }  
  
    public int precio() {  
        return 90;  
    }  
}
```

```
public class CafeLargo implements Cafe {  
    public String descripcion() {  
        return "cafe largo";  
    }  
  
    public int precio() {  
        return 100;  
    }  
}
```

...

Problema: para cada café podemos servirlo con leche (+20 centimos)

~~Duplicar clases ?~~

~~CafeCorto, CafeCortoConLeche, CafeLargo, CafeLargoConLeche ...~~

Si es común a todos los cafes que puedan llevar leche o no ... lo añadimos al tipo café



```
public interface Cafe {  
    String descripcion();  
    int precio();  
    boolean conLeche();  
    void añadeLeche();  
}
```

Podríamos seguir con la interface y añadir métodos para determinar si tiene leche o no

¡ Hay que modificar todas las subclases !

Hacer Cafe clase abstracta:

```
public abstract class Cafe {  
    private boolean conLeche;  
  
    public abstract String descripcion();  
  
    public int precio() {  
        return conLeche? 20:0;  
    }  
  
    boolean conLeche(){  
        return conLeche;  
    }  
  
    void añadeLeche() {  
        conLeche = true;  
    }  
}
```




El método en Cafe devuelve solo el precio de la leche

Café corto añade "con leche" si corresponde



```
public class CafeCorto extends Cafe {  
  
    public String descripcion() {  
        return "cafe corto " + (conLeche()? "con leche ":"");  
    }  
  
    public int precio() {  
        return 90 + super.precio();  
    }  
}
```



En la subclase sumamos al precio base

Problema: podemos servir el café cortado (+10 centimos)

...

y añadir cacao (+10 centimos)

y canela (+20 centimos)

y nata (+20 centimos)

Necesitamos una forma genérica de añadir ingredientes adicionales


```
public interface Extra {  
  
    String getNombre();  
  
    int getSuplemento();  
}
```

Cada ingrediente tendrá
una clase específica

```
public class Leche implements Extra {  
  
    @Override  
    public String getNombre() {  
        return "con leche";  
    }  
  
    @Override  
    public int getSuplemento() {  
        return 20;  
    }  
}
```



```

public abstract class Cafe {

    private List<Extra> extras = new ArrayList<>();

    public void add (Extra extra) {
        extras.add(extra);
    }

    public String descripcion() {
        StringBuilder sb = new StringBuilder();

        for(Extra extra: extras) {
            sb.append(",").append(extra.getNombre());
        }

        return sb.toString();
    }

    public abstract int precio();

    public int precioTotal() {

        int precioTotal = precio();
        for (Extra extra: extras) {
            precioTotal += extra.getSuplemento();
        }
        return precioTotal;
    }
}

```

En la clase base

← Se guardan los extras

Se consideran los extras para el nombre
y para el precio

```

public class CafeCorto extends Cafe {

    public String descripcion() {
        return "cafe corto " + super.descripcion();
    }

    public int precio() {
        return 90;
    }
}

```


Aceptable ...

pero con inconvenientes:

- La clase cafe ... algo complicada, no ?
- Es normal que CafeCorto tenga un add(extra) ?
- o un precio y un precioTotal
- y si añaden otras posibles modificaciones?
- p.e. Si el café es helado $\text{precio} = \text{precio} * 0.2$
- o cafe en terraza (no es un ingrediente extra)

Pista: ¿si a un café corto le añado leche, es un café corto o es otra cosa (objeto) ?

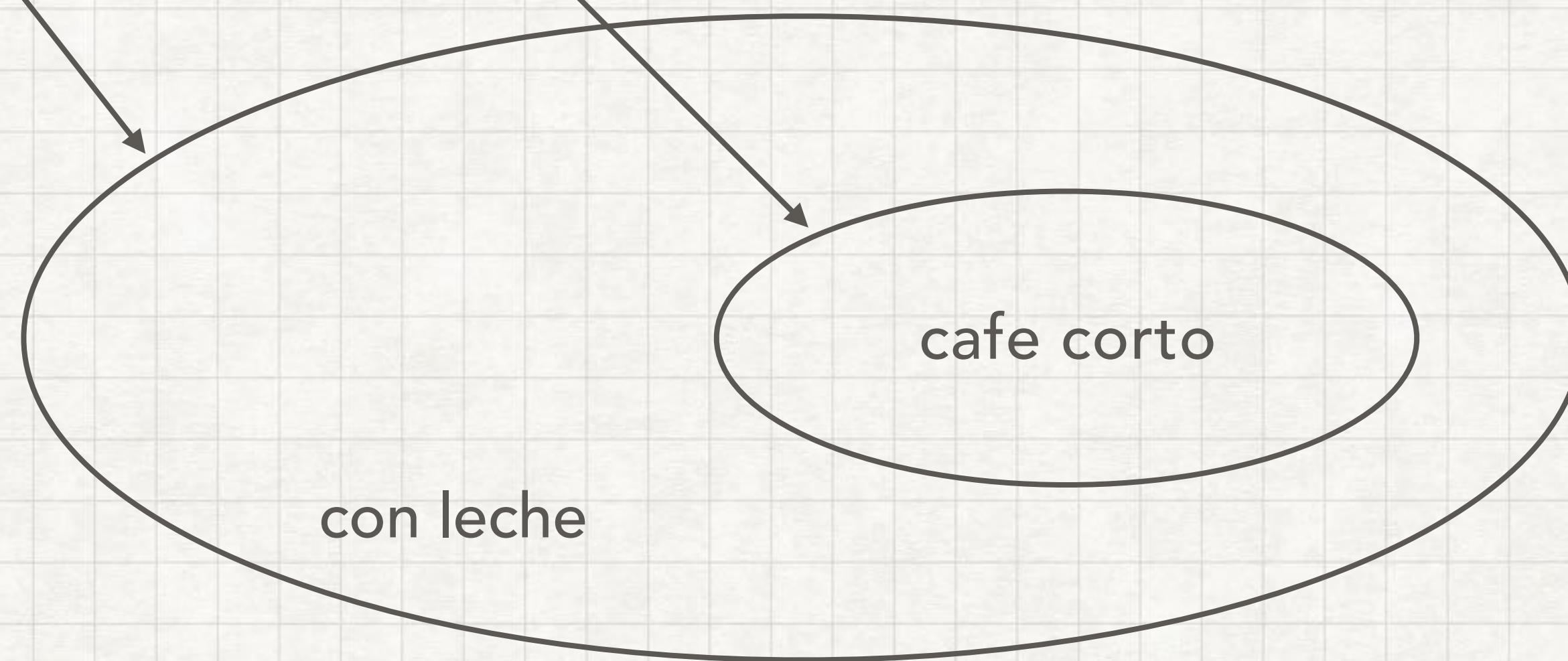
“

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

— *GoF*

”


```
new ConLeche(new CafeCorto());
```



Implementación de "con leche":
añade " con leche" a la descripción del objeto contenido
añada 20 céntimos al precio del objeto contenido


```
public interface Cafe {  
    public String descripcion();  
    public int precio();  
}
```

```
public class CafeCorto implements Cafe {  
    public String descripcion() {  
        return "cafe corto";  
    }  
  
    public int precio() {  
        return 90;  
    }  
}
```

```
new ConCacao(new ConLeche(new CafeCorto()));
```

```
public class ConLeche implements Cafe {
```

```
    private Cafe cafe;
```

Envoltorio

```
    public ConLeche(Cafe cafe) {  
        this.cafe = cafe;  
    }
```

```
@Override
```

```
    public String descripcion() {  
        return cafe.descripcion() + " con leche";  
    }
```

```
@Override
```

```
    public int precio() {  
        return cafe.precio() + 20;  
    }
```

```
}
```


DECORATOR PATTERN