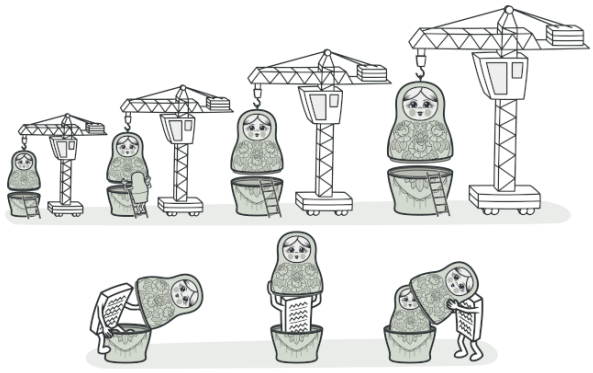
PATRONES DE DISEÑO

DECORATOR

En el caso de este patrón se basa en añadir cada vez mas funcionalidades a objetos sobre objetos es decir que dependiendo de lo que vayamos necesitando podemos ir agregándole aquella funcionalidad que se requiera con el fin de extender dicho patrón, sin embargo como tal estas funcionalidades ya no son parte del patrón porque cambiarían la herencia que lleva a cabo, de igual forma cuando no necesitemos de dicha funcionalidad porque por tal motivo ya no nos sirve pues podemos quitarla cuando deseemos.



EJEMPLO

Una aplicación de gestión de ventas de hamburguesas

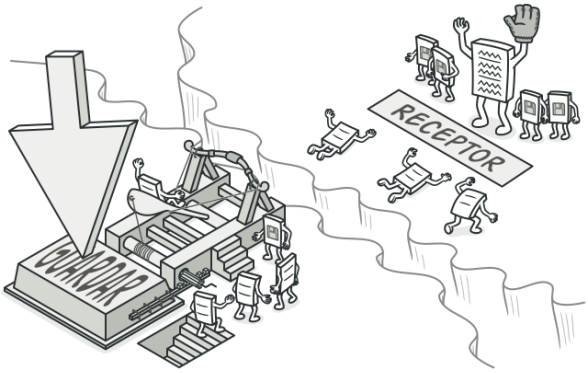
```
plo2.java ●   Ejemplo1.java ●
mplo1.java
package decorator;

public interface Hamburguer {
    public String getDescription();
    public double getPrice();
}
```

```
public abstract class ToppingDecorator implements Hamburguer {

    protected Hamburguer temporalHamburguer;

    public ToppingDecorator(Hamburguer temporalHamburguer) {
        this.temporalHamburguer = temporalHamburguer;
    }

    public String getDescription() {
        return temporalHamburguer.getDescription();
    }

    public double getPrice() {
        return temporalHamburguer.getPrice();
    }
}
```

```
public class chef {

    public static void main(String[] args) {

        Hamburguer hamburguer = new Mozzarella(new TomatoSauce(new PlainHamburguer()));
        System.out.println("Ingredients: " + hamburguer.getDescription());
        System.out.println("Total Price: " + hamburguer.getPrice());

    }

}
```

COMMAND

Este patrón como tal tiene la funcionalidad de definir ciertas solicitudes por medio de parámetros lo cual ayuda a que por medio de ese vínculo que produce el comando se pueda realizar gran cantidad de tareas que requiera la compañía, ya que reduce la cantidad de tiempo que se utiliza al realizar tantos comandos para cada cosa por aparte en cambio esta solicitud permanecerá en espera hasta que se vaya dando la respuesta a dicha solicitud, ya que hay un receptor que tomara solicitud por solicitud para dar la respectiva respuesta.



EJEMPLO

Una aplicación para el manejo de dispositivos electrónicos

mplo2.java ● Ejemplo1.java ●

mplo2.java

```java
package command

public interface Command {
    public void execute();
    public void undo();
}
```

```java
package command;

import command;

public class TurnOffAllDevices implements Command {

    List<ElectronicDevice> allDevices;

    public TurnOffAllDevices(List<ElectronicDevice> newDevices) {
        allDevices = newDevices;
    }

    public void execute() {
        for (ElectronicDevice device : allDevices) {
            device.off();
        }
    }

    public void undo() {
        for (ElectronicDevice device : allDevices) {
            device.on();
        }
    }
}
```

```java
package command;

import command;

public class DeviceButton {

    private Command command;

    public DeviceButton(Command command){
        this.command = command;
    }

    public void press(){
        command.execute();
    }

    public void pressUndo(){
        command.undo();
    }
}
```