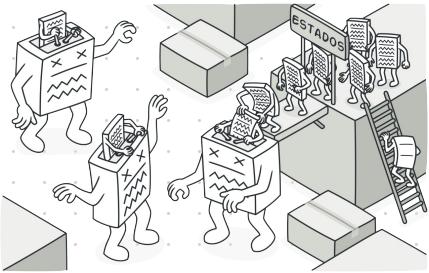


T1

IV

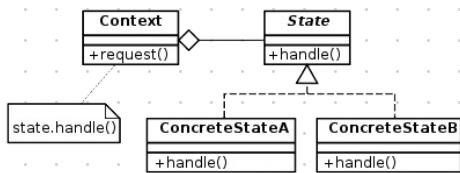
# PATRONES DE CREACION

## Patrón STATE



- Permite al objeto cambiar su **comportamiento** cuando su estado interno cambia

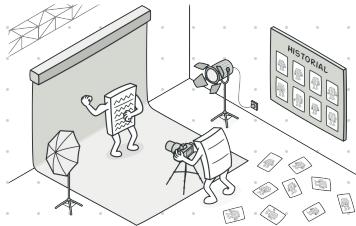
pulsar El Movi()   
 [Pantalla apagada] → desbloquea  
 [Pantalla encendida] → fijas  
 [Móvil cargando] → % de batería



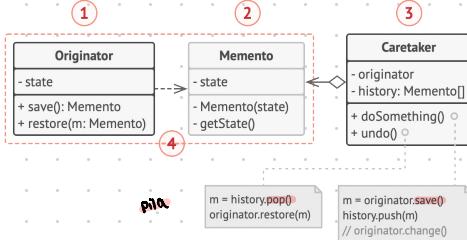
Cumple con:

- Principio Responsabilidad única  
organiza el código relacionado con estados particulares en clases separadas
- Principio abierto/cerrado  
nuevos estados → sin romper las existentes o contexto

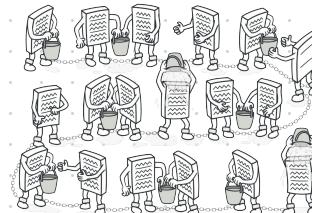
## Patrón Memento



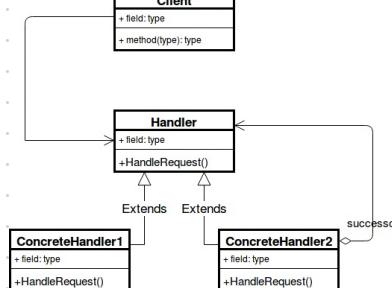
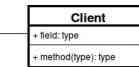
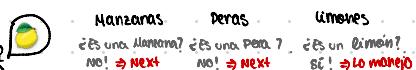
- Te permite **guardar** y **restaurar** el estado previo del obj  $\Rightarrow$  sin revelar su implementación
- Delega creación de la "instantánea"  $\Rightarrow$  Propietario del ese estado **Objeto Originador**
- Almacenamiento  $\Rightarrow$  obj especial **Memento**  
contenido  $\Rightarrow$  no accesible para otros obj q no sea el q lo produjo
- caretaker**  $\Rightarrow$  sabe "cuando" y "porque" capturar el estado original  
sabe cuando debe restaurarse



## Patrón Chain of Responsibility



Permite Pasar Solicitudes a lo largo de una manejadora.



ventajas :

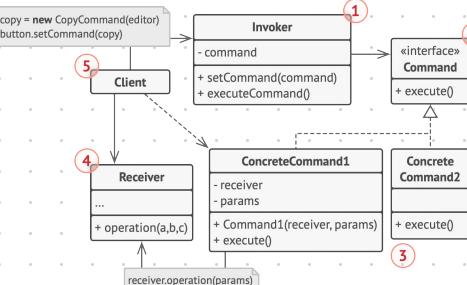
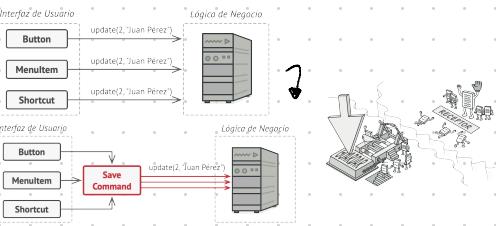
- Principio Responsabilidad única
- Principio Abierto/Cerrado
- Puedes controlar orden de control de solicitudes

# PATRONES DE CREACION

## Patrón Command

Convierte solicitud Obj indep q contiene toda la info solicitud

- Permite parametrizar métodos con soluciones
- retroceder/poner otra ejecución de una solicitud
- soportar operaciones q no se pueden realizar



Emitida / Invocadora: inicializar solicitudes (1)

Command: declara un único método para ejecutar comando (2)

Comandos concretos: Implementa varios tipos de solicitudes (3)

Receptor: Contiene cierta lógica de Negocio (4)

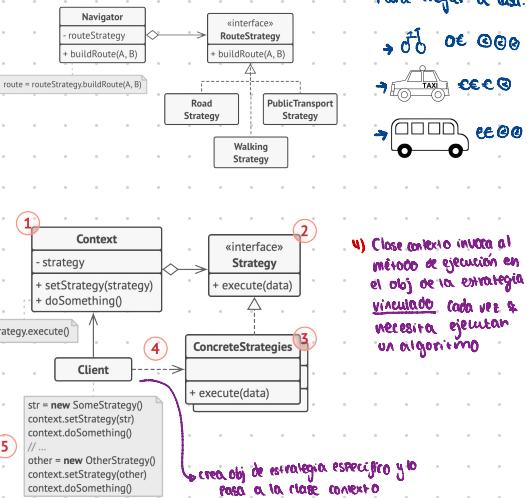
Cliente: Crea y Config. obj del comando (5)

## Patrón Strategy

comportamiento

Permite definir una familia de algoritmos cada uno en una clase separada. hacer sus obj intercambiables

Ejemplo: APP de rutas



### Ventajas

- puedes intercambiar algoritmos usados dentro de un obj durante el de ejecución
- listar detalles de implementación del algoritmo
- puedes sustituir herencia composición
- principio abierto/cerrado  
(+ estrategias + sin tener q cambiar) el contexto

### Inconvenientes

- clientes deben conocer estrategias
- Muchas lenguajes modernos envs de obj de estrategia
- 2 algoritmos q normalmente cambian no conviven

- 5) debe crearse con instancias de una clase heredada concreta particular cliente solicita una

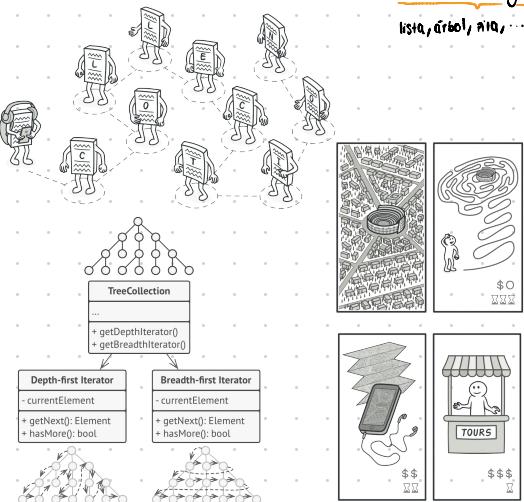
"¿Dónde está el resto de la colección?"

los demás no son fundamentales para el patrón. Por eso los demás los testé en la más alta fase

## Patrón Iterator

comportamiento

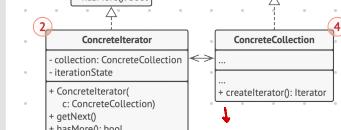
Permite recorrer elementos de una colección sin exponer su representación Subyacente



1) declara operaciones necesarias para recorrer una colección  
[Entrar, sig elem., revisar iter,...]

2) Algoritmos específicos para recorrer una colección

3) Declara 1 o varios obj  
Obtener iteradores compatibles con la colección



- 4) devuelven nuevas instancias de una clase heredada concreta particular

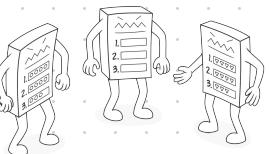
"¿Dónde está el resto de la colección?"

los demás no son fundamentales para el patrón. Por eso los demás los testé en la más alta fase

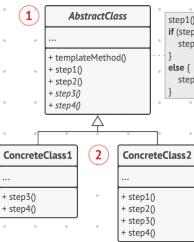
# PATRONES DE COMPORTAMIENTO

## Template Method

Define el esqueleto de un algoritmo en la superclase pero permite que las subclases sobreescriban pasos del algoritmo sin romper estructura.



Estructura:



1) Opción:  
↳ métodos → pasos de un algoritmo  
↳ método puntoilla q invoca orden

2) Pueden sobreescritir pasos  
↳ el propio método puntoilla

### Inconvenientes

- Puede violarse principio de sustitución liskov
- Exprimiendo una implementación defecto de un paso a través de una subclase

### Ventajas

Permitir Cliente → sobreescribir pt algoritmo  
Evitar código duplicado dentro subclase

## Patrón Mediator

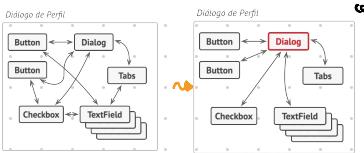
Permite ✓ las dependencias covariantes entre objetos

El Patrón restringe → comunicaciones directas entre los obj.  
[restando a colaborar solo a mediador]



↓ solución?  
↳ detengo → comunicación directa entre los componentes q quieren  
Patrón Mediator

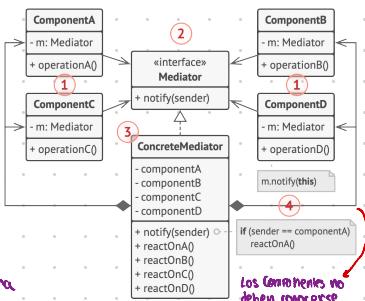
↳ Componentes cobr. indirectamente → invocando obj. Mediador  
redirecciones llamadas a los componentes adecuados



2) Interfaz Mediadora: declara métodos de comunicación con los componentes (normalmente incluye método de notificación)

3) Encapsula relaciones entre valores componentes

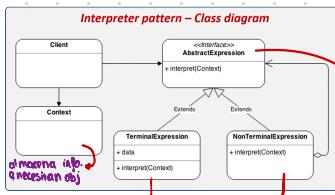
4) Varios clases q contienen una logica de negocio  
↳ comp → reg. integrar Mediadora



Los componentes no deben conocerse  
entre sí → problema → mediador

## Patrón Interpreter

Dado un lenguaje → define representación para su gramática junto con un intérprete del lenguaje



Una sola expresión y se interpreta en un solo obj

contiene varias expresiones q serán interpretadas

expresiones regulares  
✓ Valores terminales  
✗ No terminales

Abstract Expression declara una operación abstracta de interpretar que es común a todos los nodos en el árbol de sintaxis abstracta

Ejemplo: N° Romano no decimal

Contexto: Entrada: N° Romano Salida: int

Clase Abstracción: one = " " → símbolos q terreno dig.  
four = " " → en las tablas  
five = " " → ...  
↳ tiene Método Interpretar

definimos los 4º símbolos

definimos símbolos de expresiones, 10

x = xc  
expresiones 100

c = cd  
expresiones 1000

d = dn  
expresiones 1000

