Patrones de diseño III

IX

X

2025

UNRN

Universidad Nacional
de **Río Negro**

r27

Hablemos del TPI

Trabajo Práctico Integrador

T.P.I.

1

La defensa es requisito para la promoción de la materia.

La entrega es requisito de regularización

sin excepciones

T.P.I.

2

La entrega (vía formulario) es 24hs antes del día de la defensa.

17 de junio

T.P.I.

3

Si no están en condiciones de promocionar, la defensa queda para cuando se anoten al final.

T.P.I.

4

En equipos de hasta 2.

T.P.I.

5

La defensa es
técnicamente
un parcial

T.P.I.

6

Hay solo una instancia de ‘recuperatorio’,

{cuando se anoten a rendir el final y la defensa, ahí será individual}

T.P.I.

7

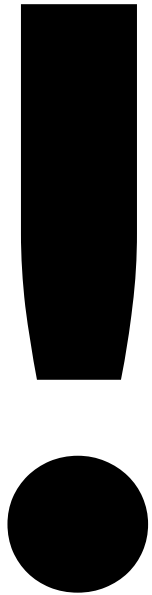
Se evaluará:

- El funcionamiento
- La aplicación de lo visto
- Las reglas de estilo
- La participación del equipo
- Tests + documentación

Abran hilo

{ Una característica de la Agenda [TP9] } que ha quedado suelta

No admite duplicados



Como `java.util.Set`

Pero require de un hashCode y equals

Patrones

Comportamiento

Comportamiento

B se enfocan en la interacción y distribución de responsabilidades entre objetos.

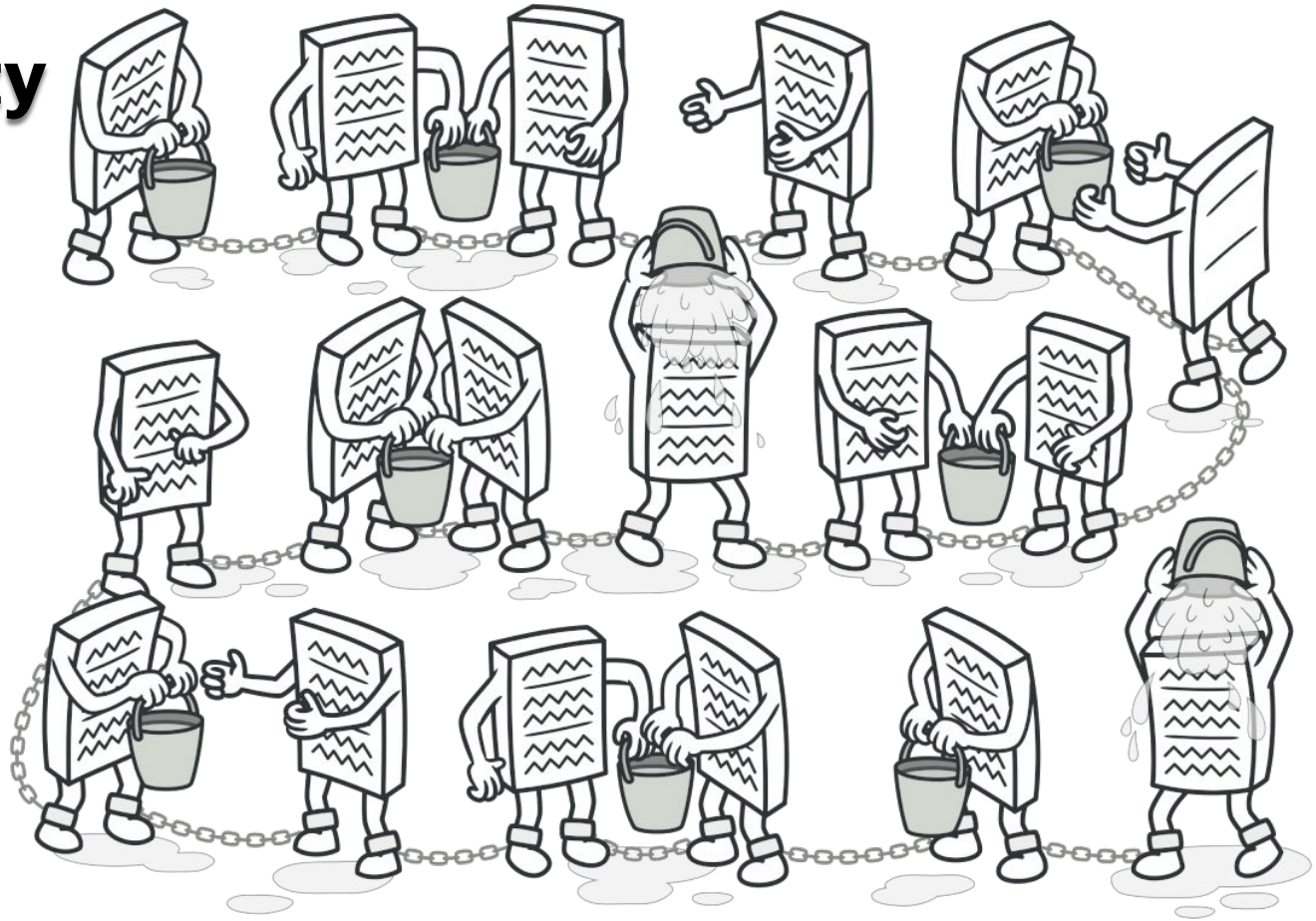
Comportamiento

- Chain of Responsability
- Command
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

Comportamiento

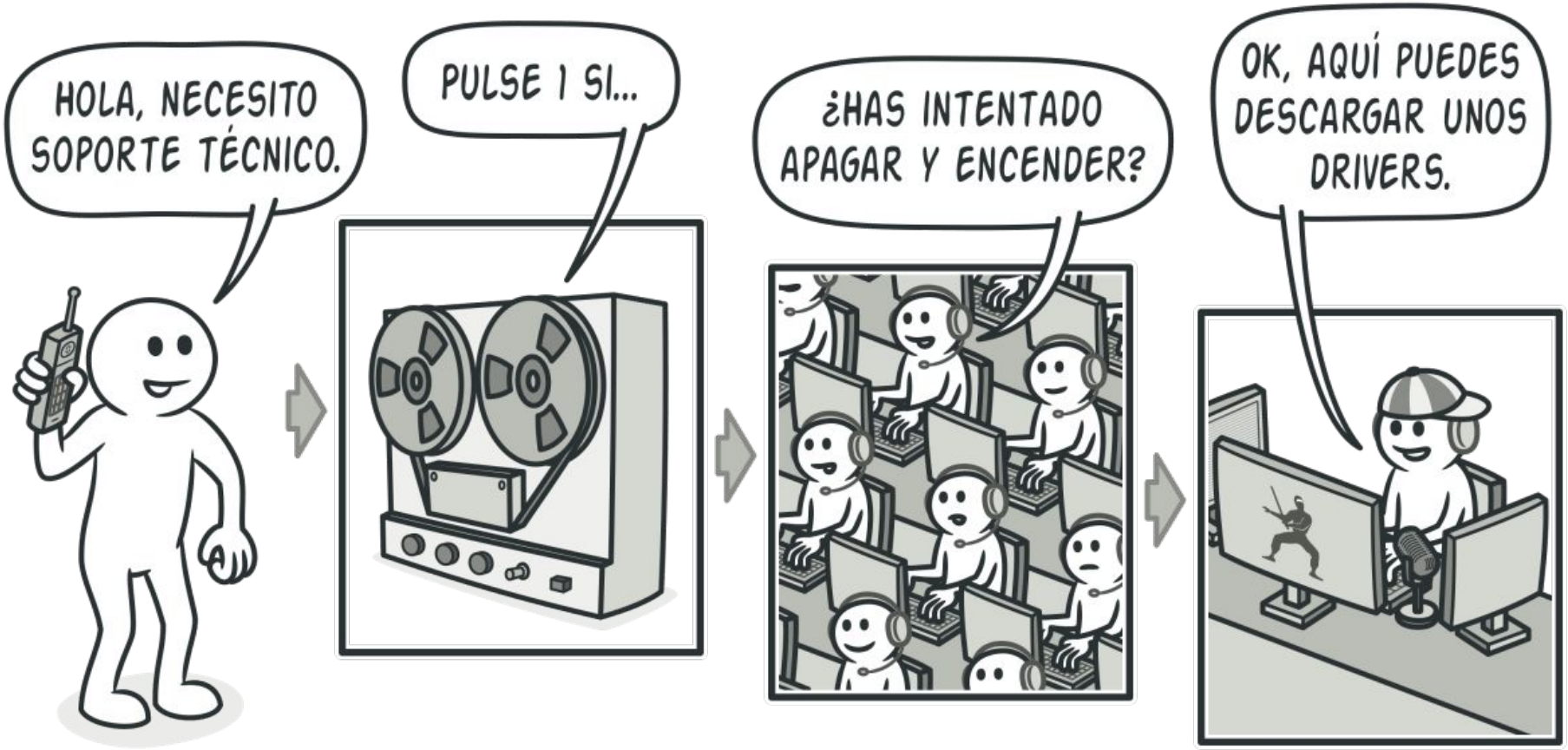

¿Preguntas?

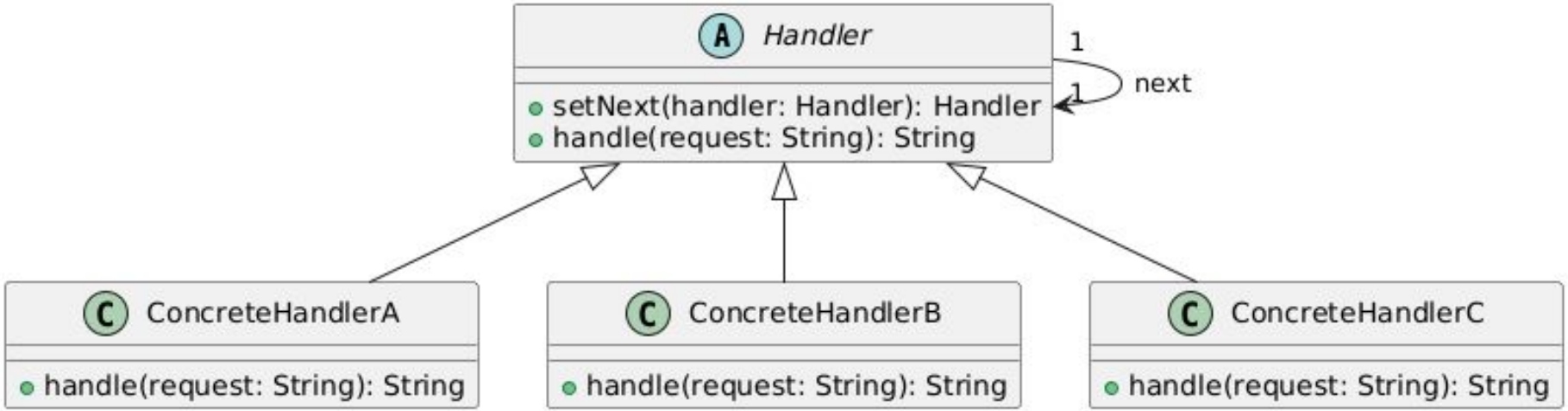
Chain of Responsibility

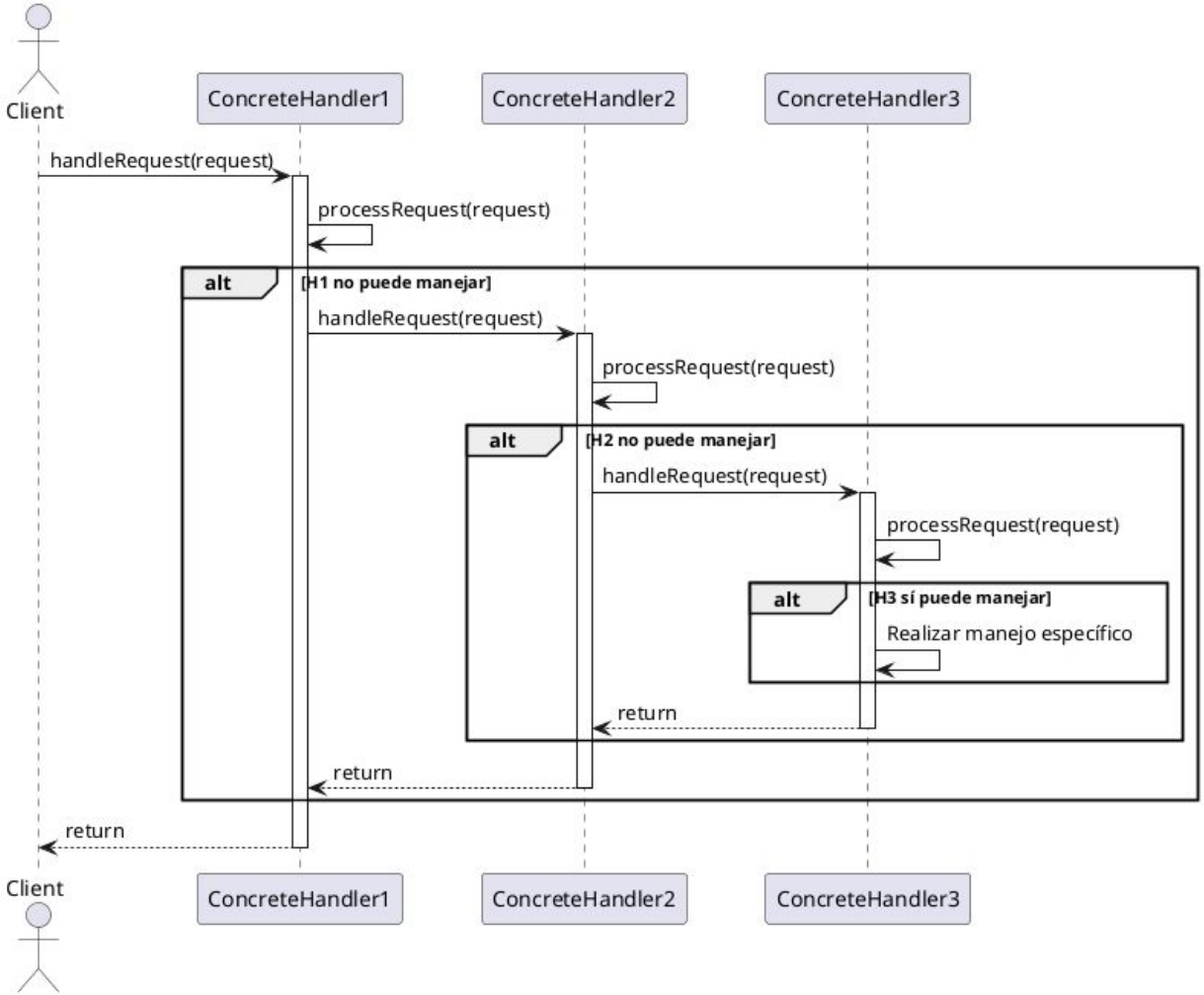


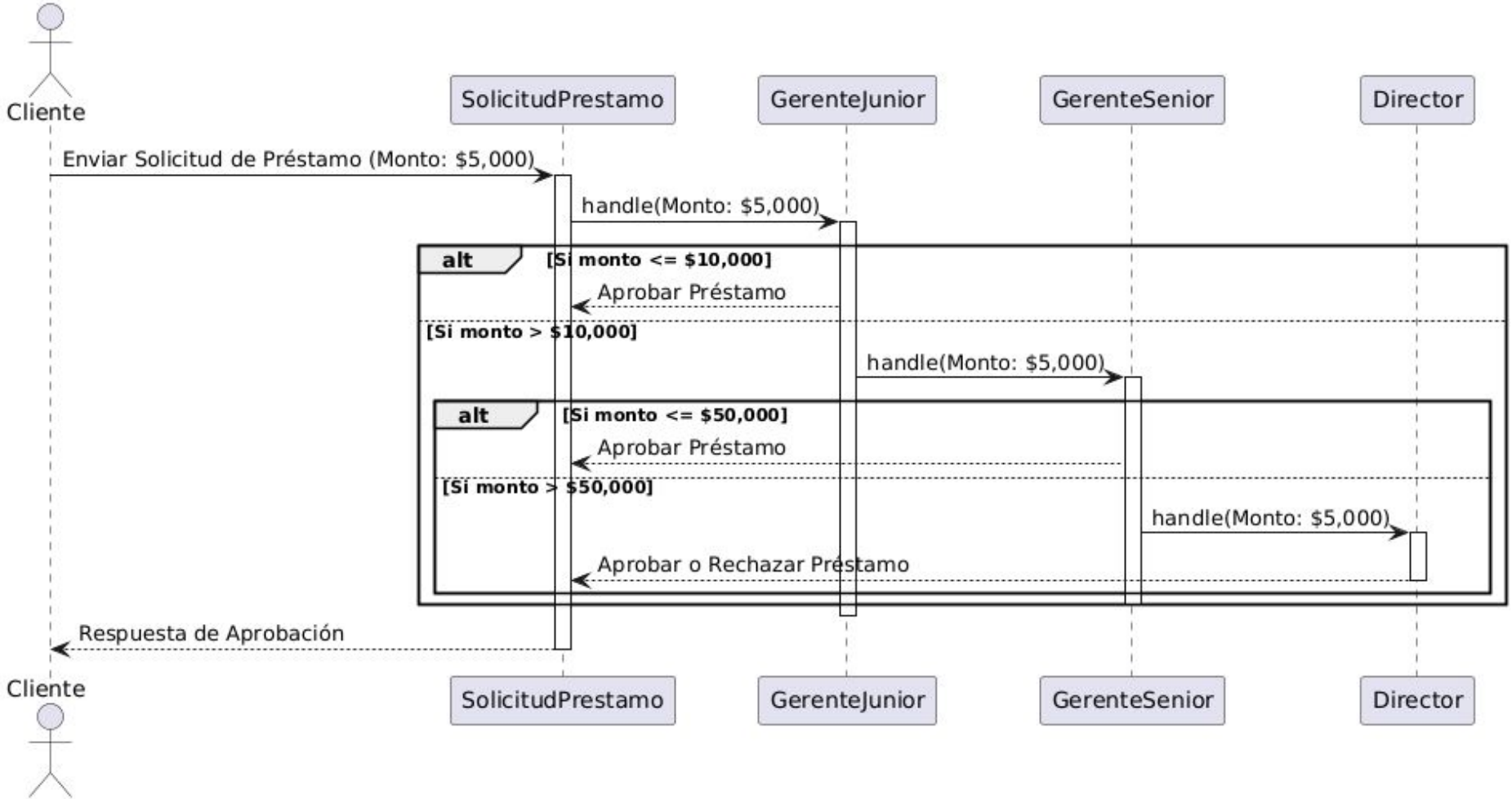
Chain of Responsibility

Pasa una solicitud a lo largo de una cadena de posibles responsables hasta que uno de ellos la procesa.


HOLA, NECESITO SOPORTE TÉCNICO.
PULSE 1 SI...
¿HAS INTENTADO APAGAR Y ENCENDER?
OK, AQUÍ PUEDES DESCARGAR UNOS DRIVERS.

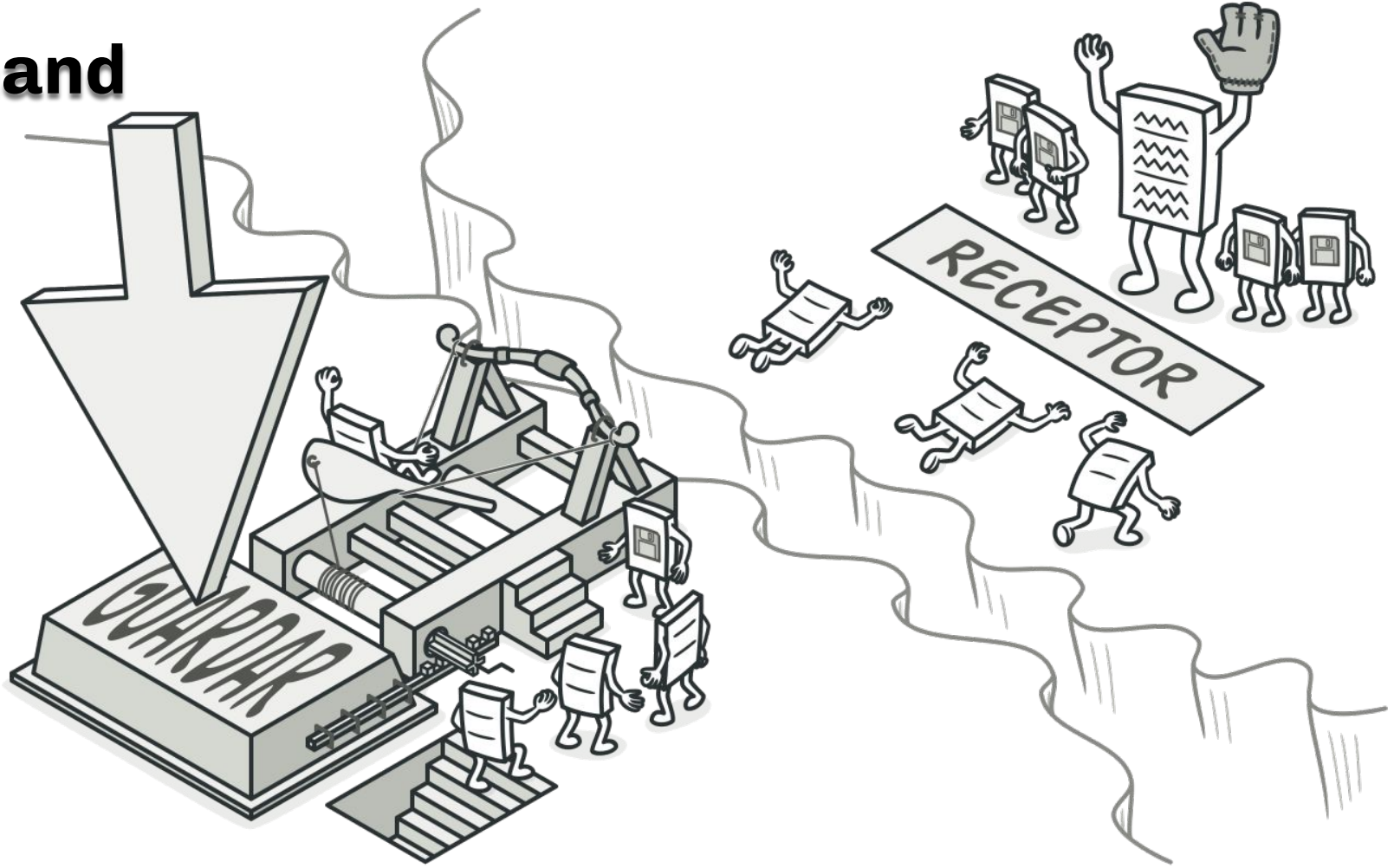

Handler
setNext(handler: Handler): Handler
handle(request: String): String
1
1
next
ConcreteHandlerA
handle(request: String): String
ConcreteHandlerB
handle(request: String): String
ConcreteHandlerC
handle(request: String): String


Client
ConcreteHandler1
ConcreteHandler2
ConcreteHandler3
handleRequest(request)
processRequest(request)
alt
[H1 no puede manejar]
handleRequest(request)
processRequest(request)
alt
[H2 no puede manejar]
handleRequest(request)
processRequest(request)
alt
[H3 sí puede manejar]
Realizar manejo específico
return
return
return
Client
ConcreteHandler1
ConcreteHandler2
ConcreteHandler3


Cliente
SolicitudPrestamo
GerenteJunior
GerenteSenior
Director
Enviar Solicitud de Préstamo (Monto: $5,000)
handle(Monto: $5,000)
alt
[Si monto <= $10,000]
Aprobar Préstamo
[Si monto > $10,000]
handle(Monto: $5,000)
alt
[Si monto <= $50,000]
Aprobar Préstamo
[Si monto > $50,000]
handle(Monto: $5,000)
Aprobar o Rechazar Préstamo
Respuesta de Aprobación
Cliente
SolicitudPrestamo
GerenteJunior
GerenteSenior
Director


¿Preguntas?

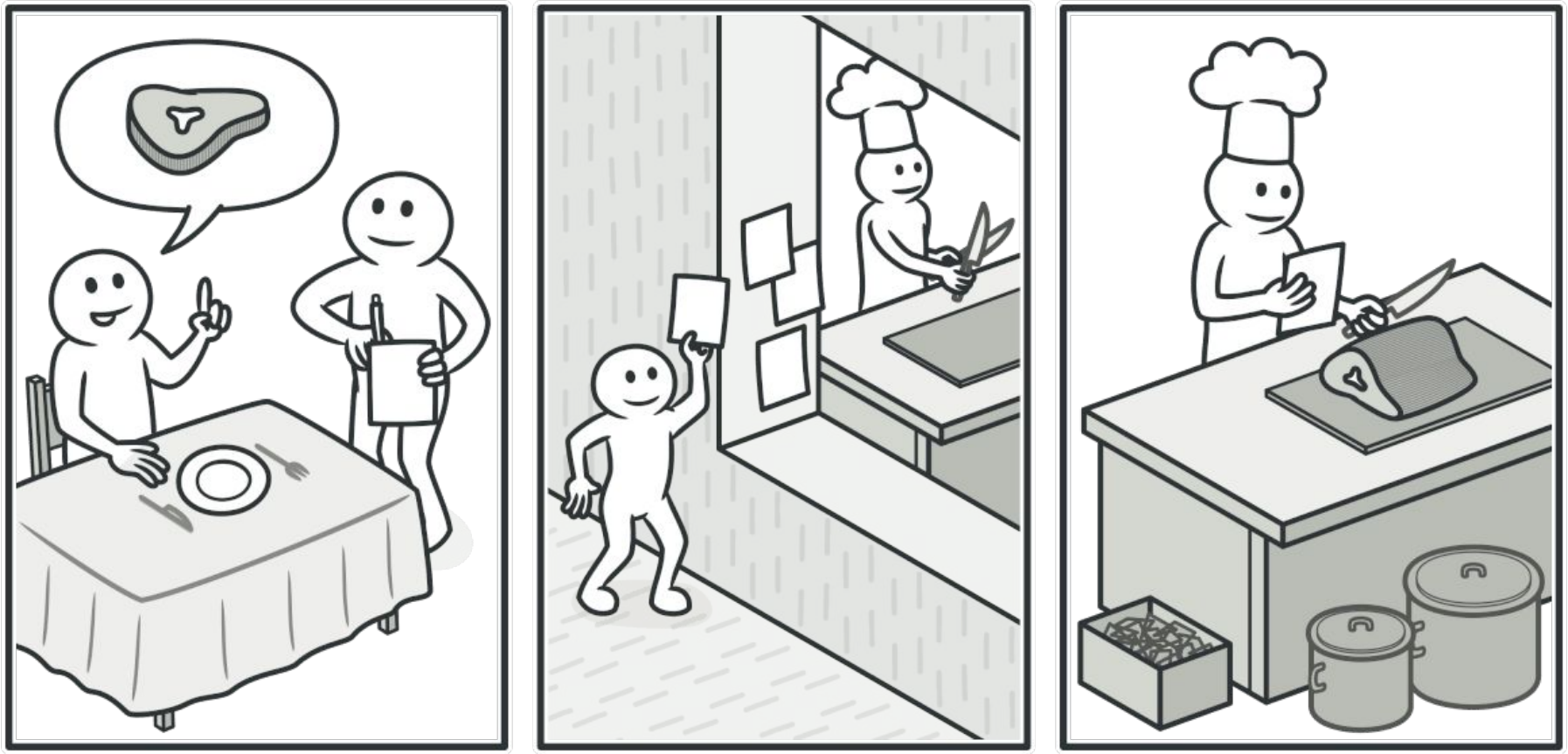
Command

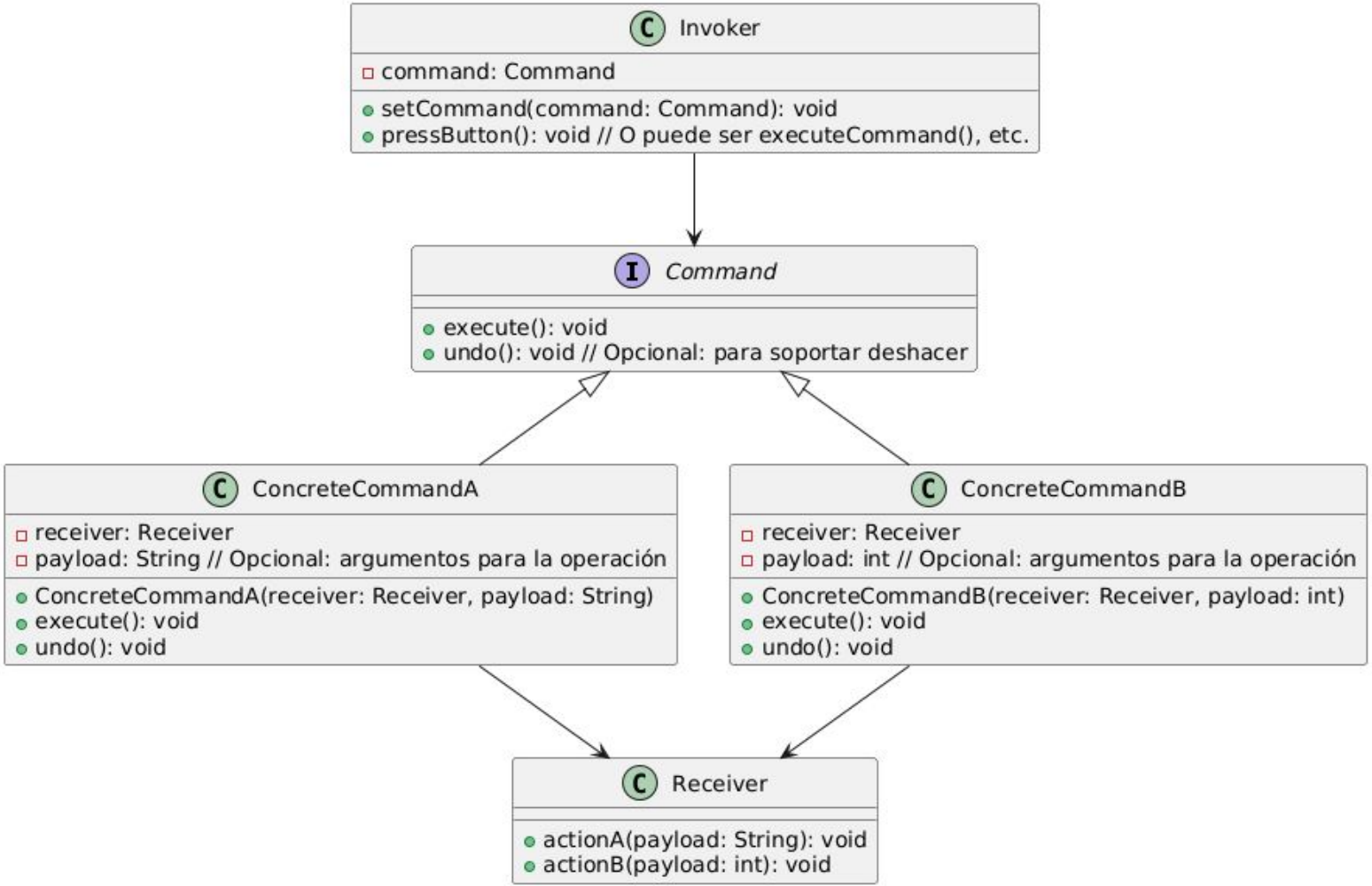


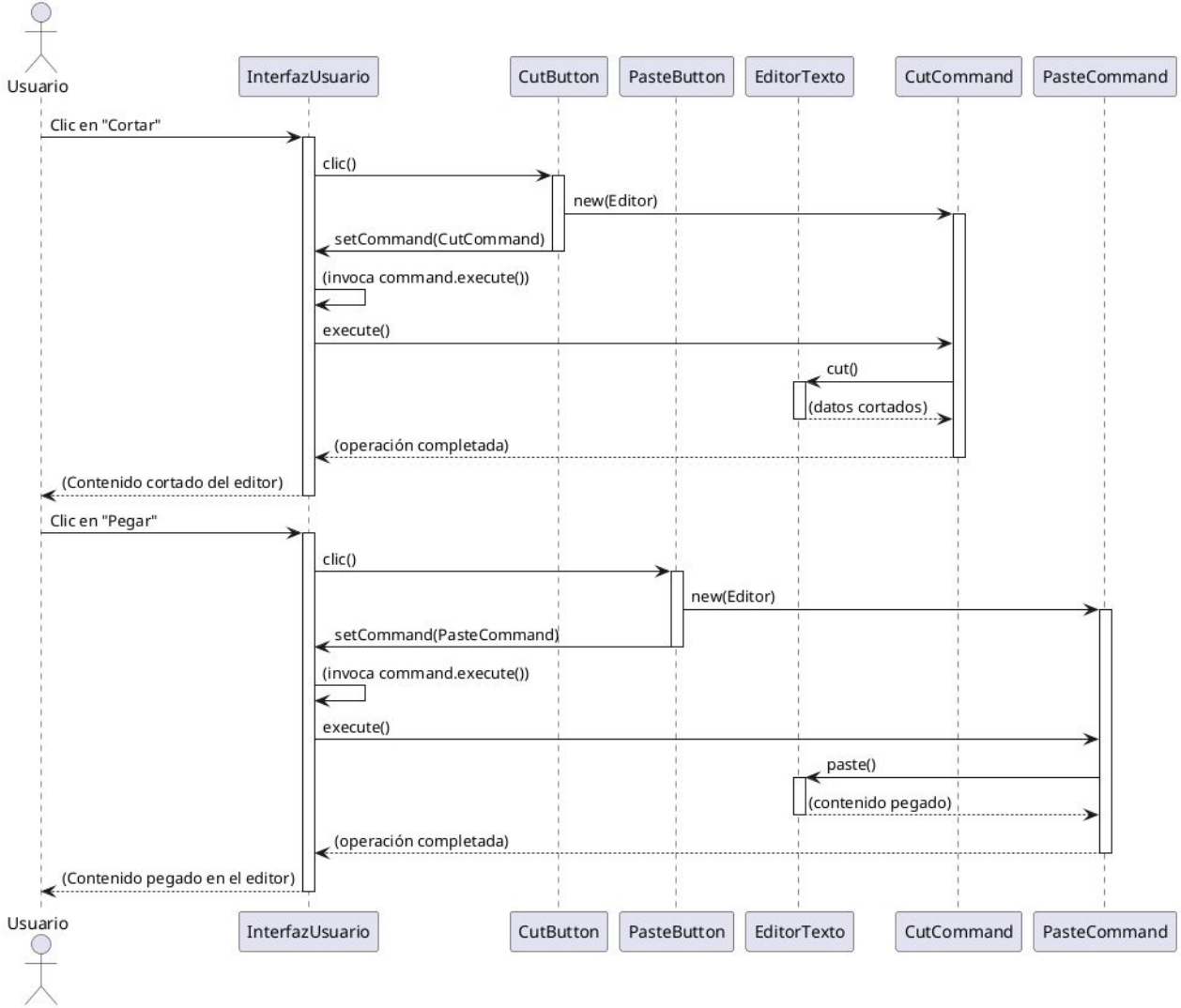
Command

transforma una solicitud en un objeto independiente que contiene toda la información necesaria.

Esta encapsulación permite: parametrizar clientes con solicitudes, poner las solicitudes en una cola o registrarlas (para deshacerlas).




Invoker
- command: Command
+ setCommand(command: Command): void
+ pressButton(): void // O puede ser executeCommand(), etc.
Command
+ execute(): void
+ undo(): void // Opcional: para soportar deshacer
ConcreteCommandA
- receiver: Receiver
- payload: String // Opcional: argumentos para la operación
+ ConcreteCommandA(receiver: Receiver, payload: String)
+ execute(): void
+ undo(): void
ConcreteCommandB
- receiver: Receiver
- payload: int // Opcional: argumentos para la operación
+ ConcreteCommandB(receiver: Receiver, payload: int)
+ execute(): void
+ undo(): void
Receiver
+ actionA(payload: String): void
+ actionB(payload: int): void


Usuario
InterfazUsuario
CutButton
PasteButton
EditorTexto
CutCommand
PasteCommand
Clic en "Cortar"
clic()
new(Editor)
setCommand(CutCommand)
(invoca command.execute())
execute()
cut()
(datos cortados)
(operación completada)
(Contenido cortado del editor)
Clic en "Pegar"
clic()
new(Editor)
setCommand(PasteCommand)
(invoca command.execute())
execute()
paste()
(contenido pegado)
(operación completada)
(Contenido pegado en el editor)
Usuario
InterfazUsuario
CutButton
PasteButton
EditorTexto
CutCommand
PasteCommand

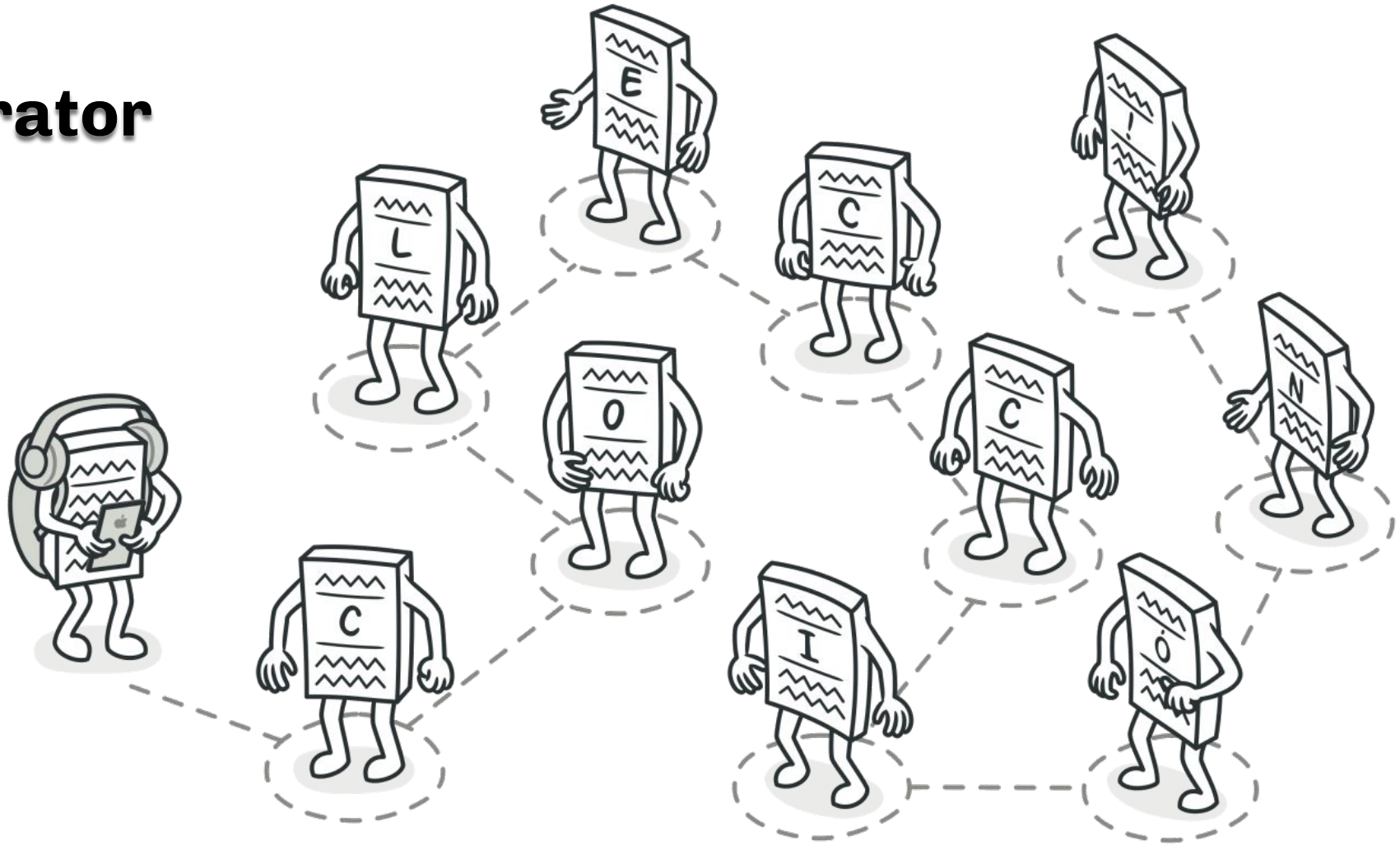
{

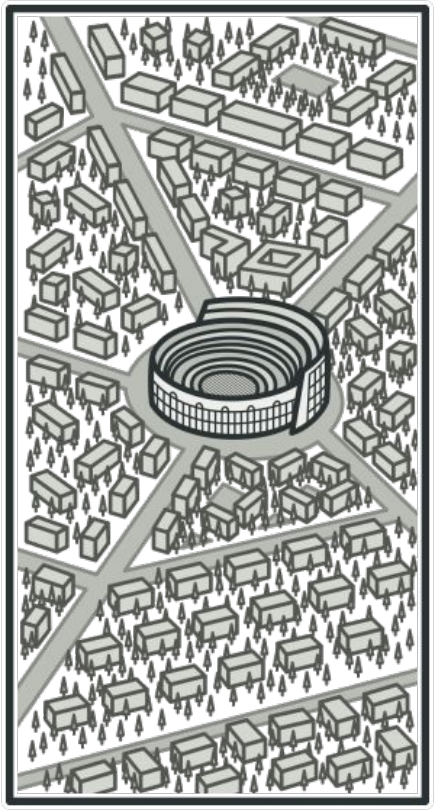
El registro de Command's permite implementar deshacer

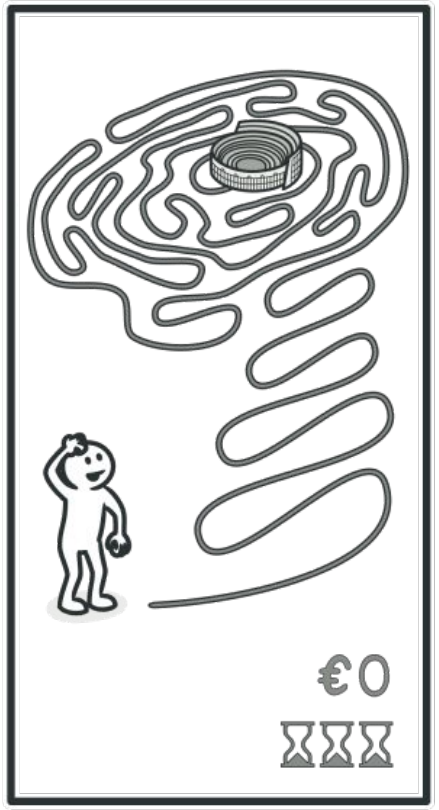
}


¿Preguntas?

Iterator





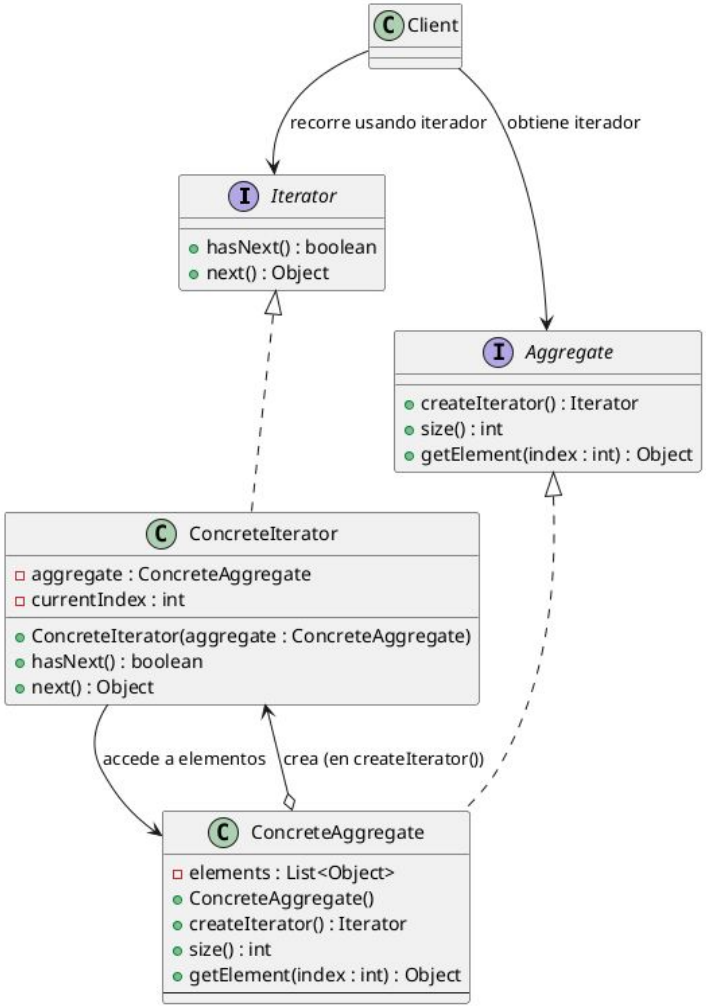

€0


€€


EXCURSIONES
€€€

Iterator

proporciona una forma de acceder a los elementos de un objeto agregado secuencialmente sin exponer su representación interna.


Client
recorre usando iterador
obtiene iterador
Iterator
+ hasNext() : boolean
+ next() : Object
Aggregate
+ createIterator() : Iterator
+ size() : int
+ getElement(index : int) : Object
ConcreteIterator
- aggregate : ConcreteAggregate
- currentIndex : int
+ ConcreteIterator(aggregate : ConcreteAggregate)
+ hasNext() : boolean
+ next() : Object
accede a elementos
crea (en createIterator())
ConcreteAggregate
- elements : List<Object>
+ ConcreteAggregate()
+ createIterator() : Iterator
+ size() : int
+ getElement(index : int) : Object

{ Ideal para las ‘vistas’ de la Agenda }

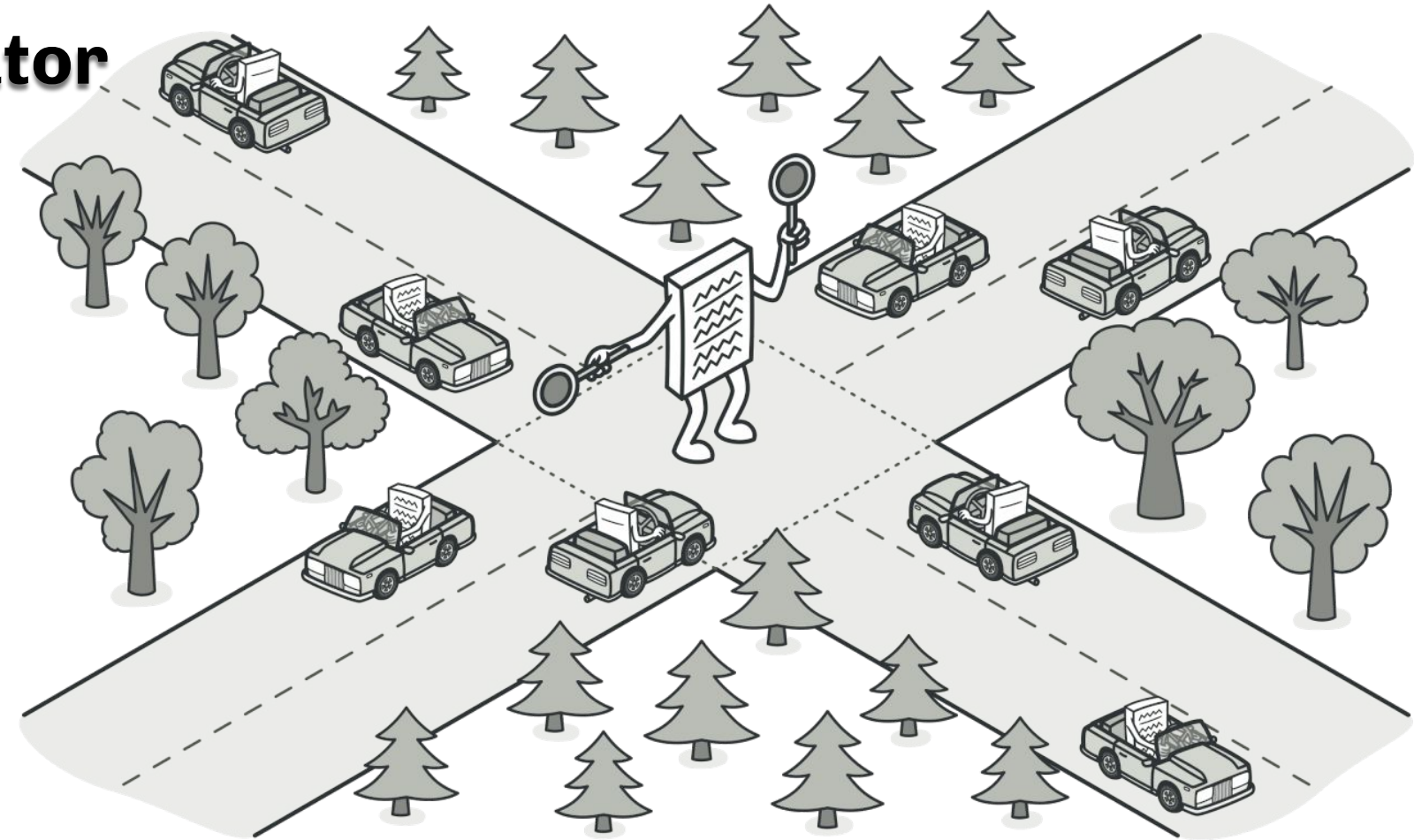
Quien necesite recorrer la agenda, *solo* recorre

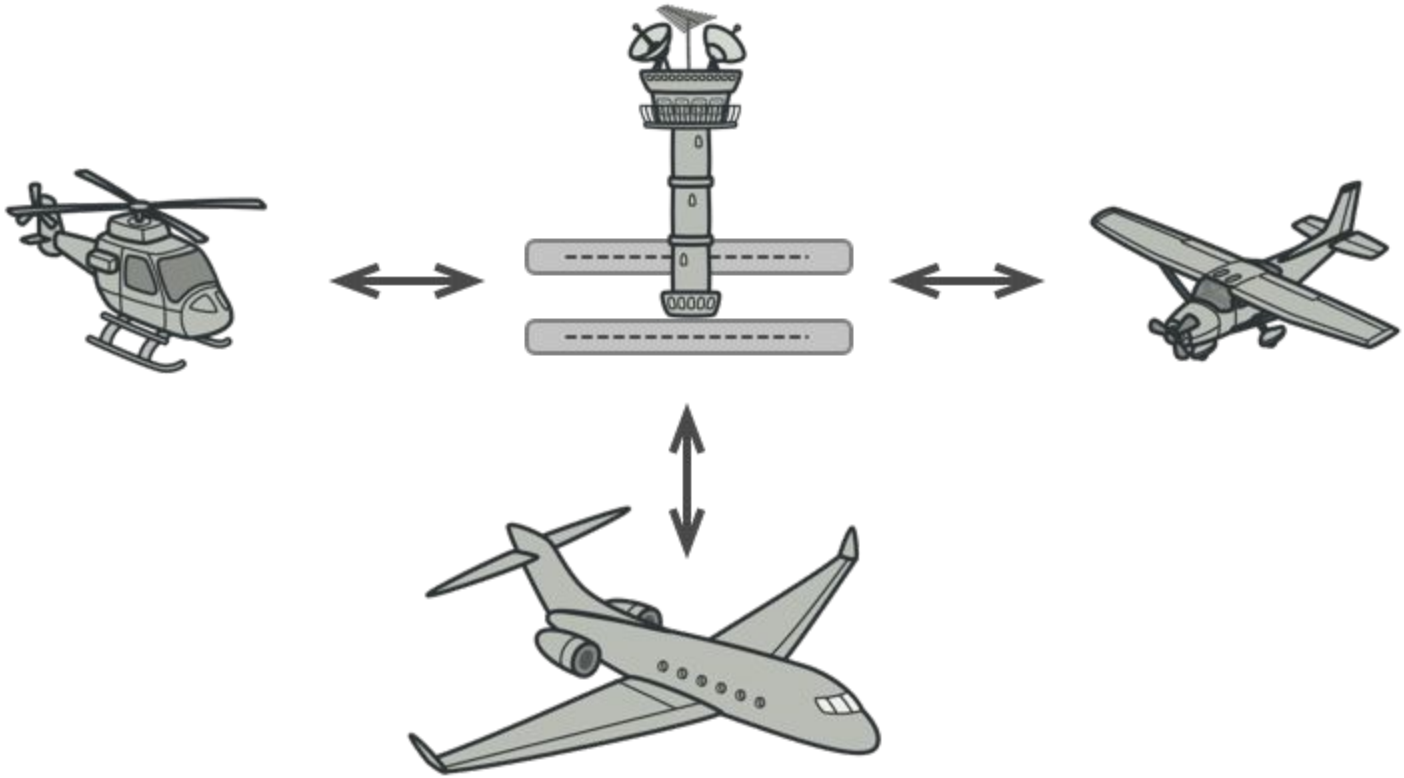
```
public class Agenda {
    …
    public Iterator<Contacto> porEmail()
    public Iterator<Contacto> porComparator(Comparator<…> c)
}

new ArrayList(porComparator()) // si realmente hace falta
```


¿Preguntas?

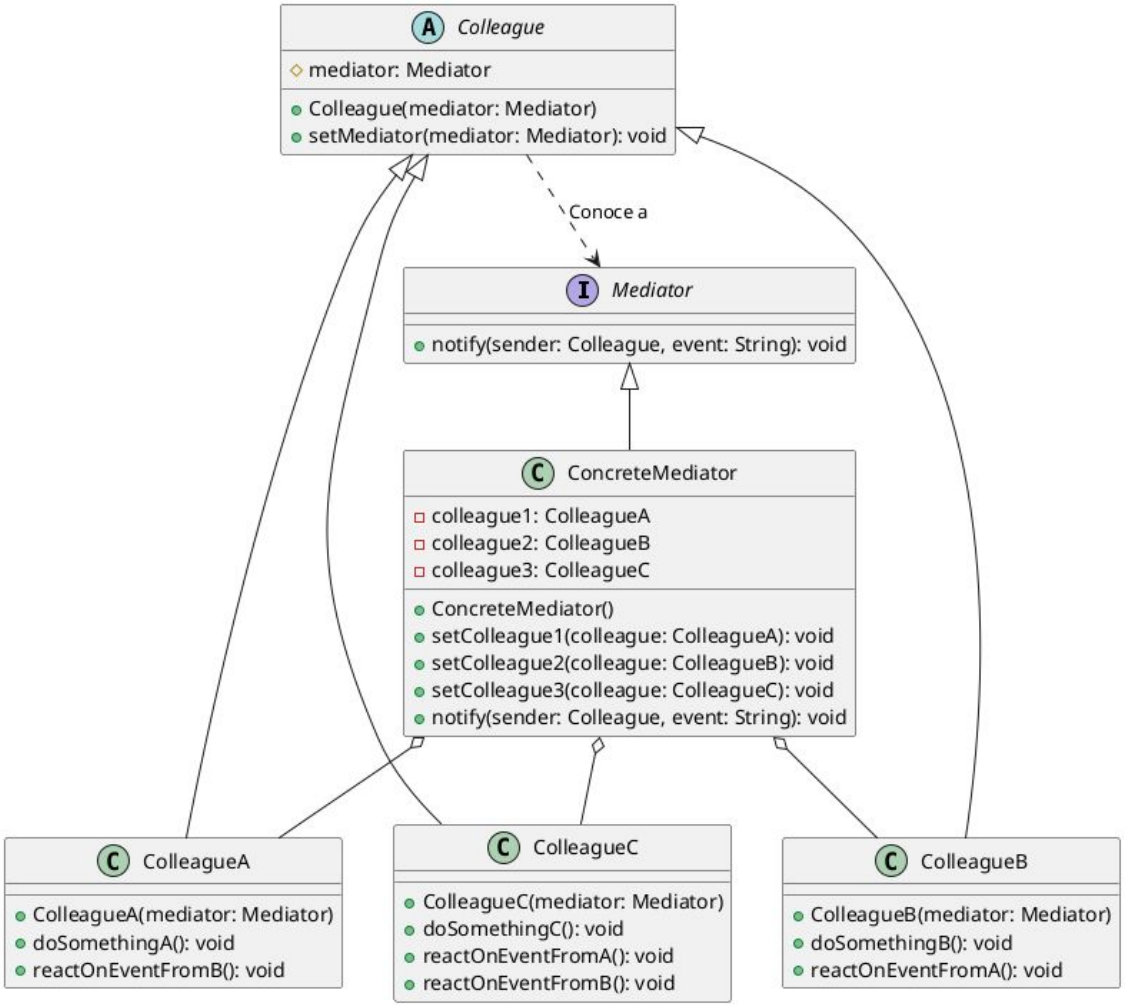
Mediator

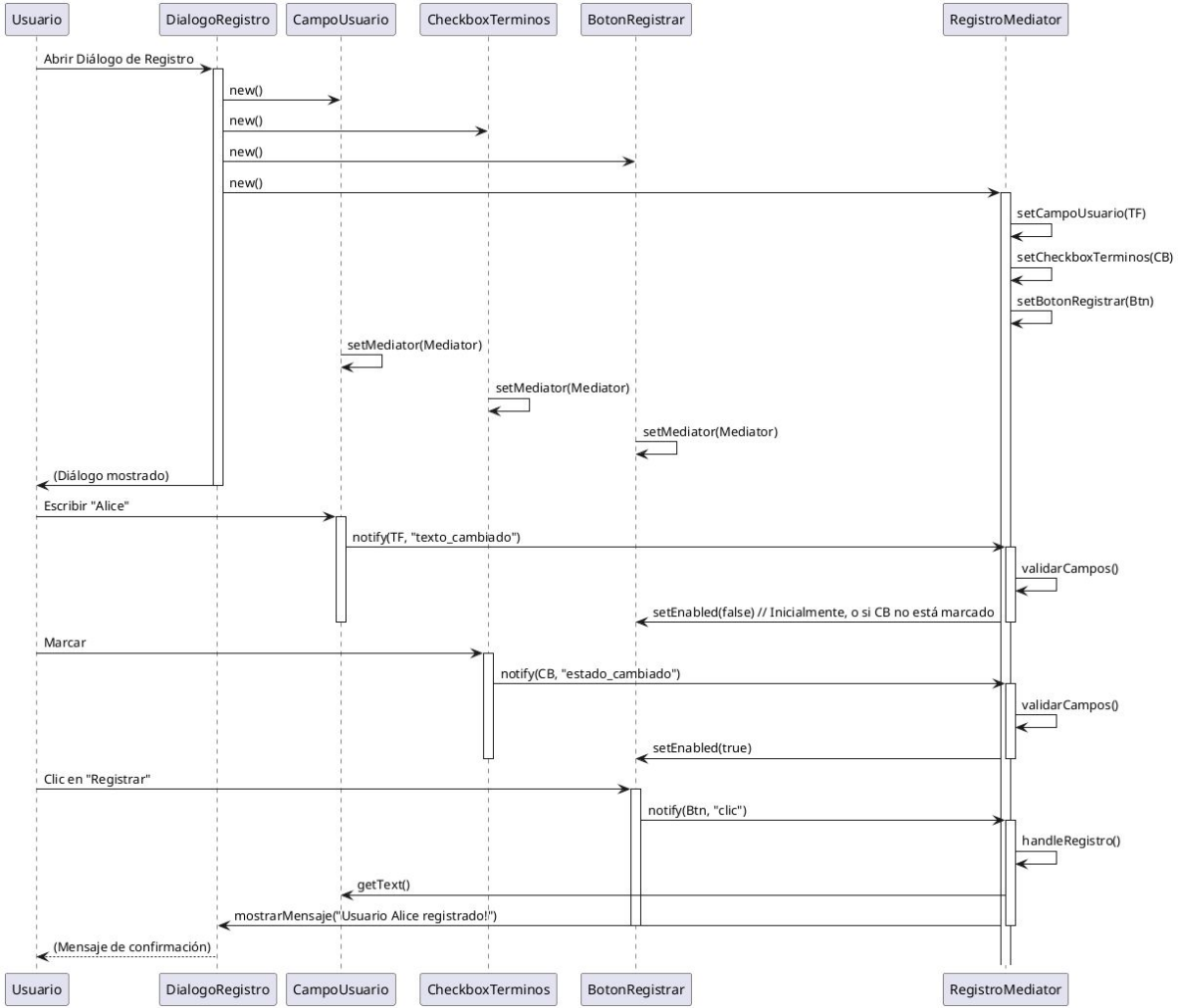




Mediator

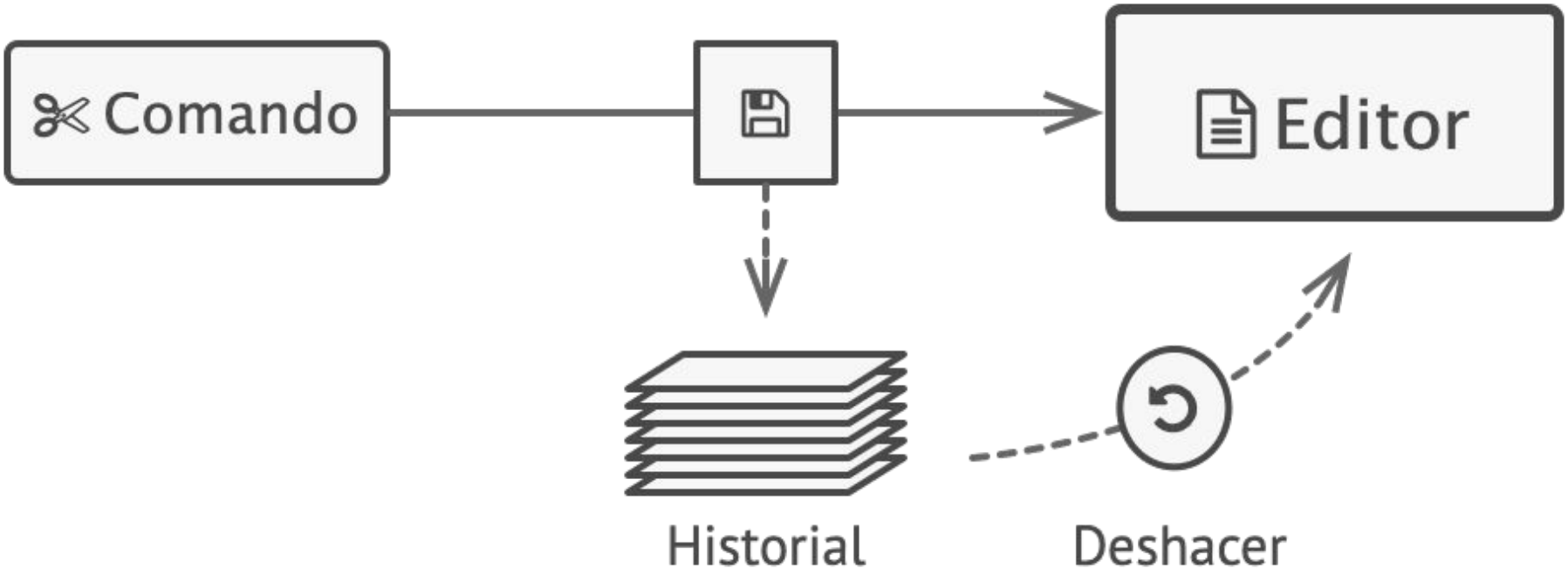
define un objeto que encapsula cómo interactúa un conjunto de objetos fomentando el acoplamiento débil al evitar que los objetos se refieran entre sí explícitamente, y te permite variar su interacción de forma independiente.


Colleague
mediator: Mediator
Colleague(mediator: Mediator)
setMediator(mediator: Mediator): void
Conoce a
Mediator
notify(sender: Colleague, event: String): void
ConcreteMediator
colleague1: ColleagueA
colleague2: ColleagueB
colleague3: ColleagueC
ConcreteMediator()
setColleague1(colleague: ColleagueA): void
setColleague2(colleague: ColleagueB): void
setColleague3(colleague: ColleagueC): void
notify(sender: Colleague, event: String): void
ColleagueA
ColleagueA(mediator: Mediator)
doSomethingA(): void
reactOnEventFromB(): void
ColleagueC
ColleagueC(mediator: Mediator)
doSomethingC(): void
reactOnEventFromA(): void
reactOnEventFromB(): void
ColleagueB
ColleagueB(mediator: Mediator)
doSomethingB(): void
reactOnEventFromA(): void


Usuario
DialogoRegistro
CampoUsuario
CheckboxTerminos
BotonRegistrar
RegistroMediator
Abrir Diálogo de Registro
new()
new()
new()
new()
setCampoUsuario(TF)
setCheckboxTerminos(CB)
setBotonRegistrar(Btn)
setMediator(Mediator)
setMediator(Mediator)
setMediator(Mediator)
(Diálogo mostrado)
Escribir "Alice"
notify(TF, "texto_cambiado")
validarCampos()
setEnabled(false) // Inicialmente, o si CB no está marcado
Marcar
notify(CB, "estado_cambiado")
validarCampos()
setEnabled(true)
Clic en "Registrar"
notify(Btn, "clic")
handleRegistro()
getText()
mostrarMensaje("Usuario Alice registrado!")
(Mensaje de confirmación)
Usuario
DialogoRegistro
CampoUsuario
CheckboxTerminos
BotonRegistrar
RegistroMediator

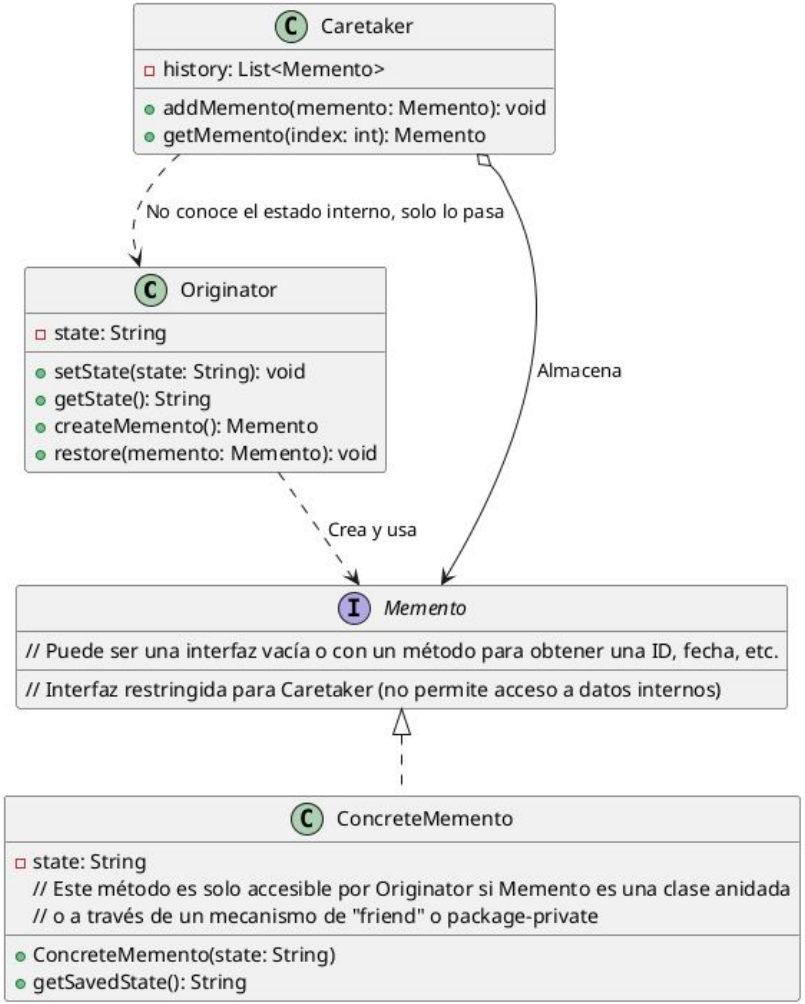

¿Preguntas?

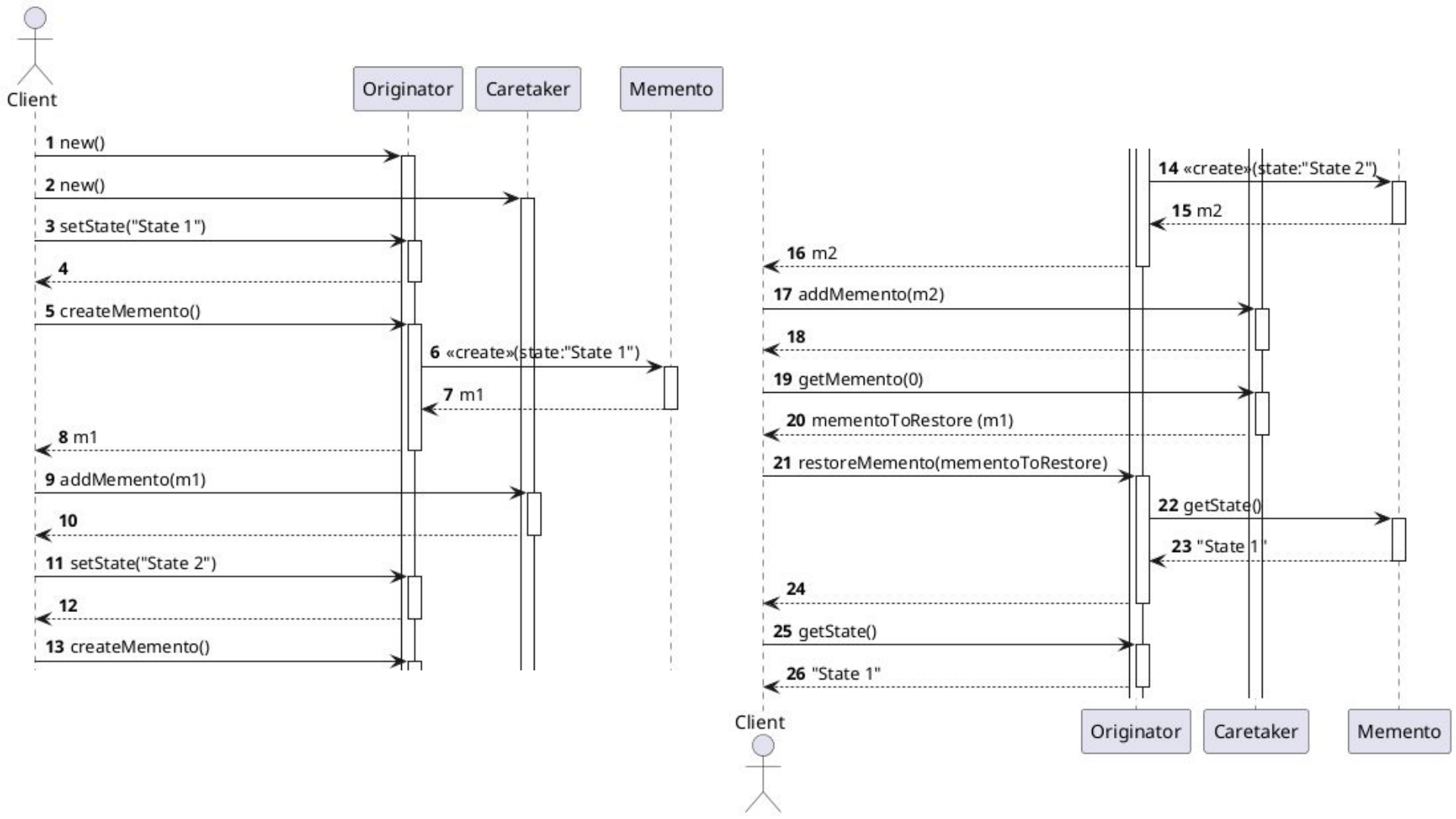
Memento


Comando
Editor
Historial
Deshacer

Memento

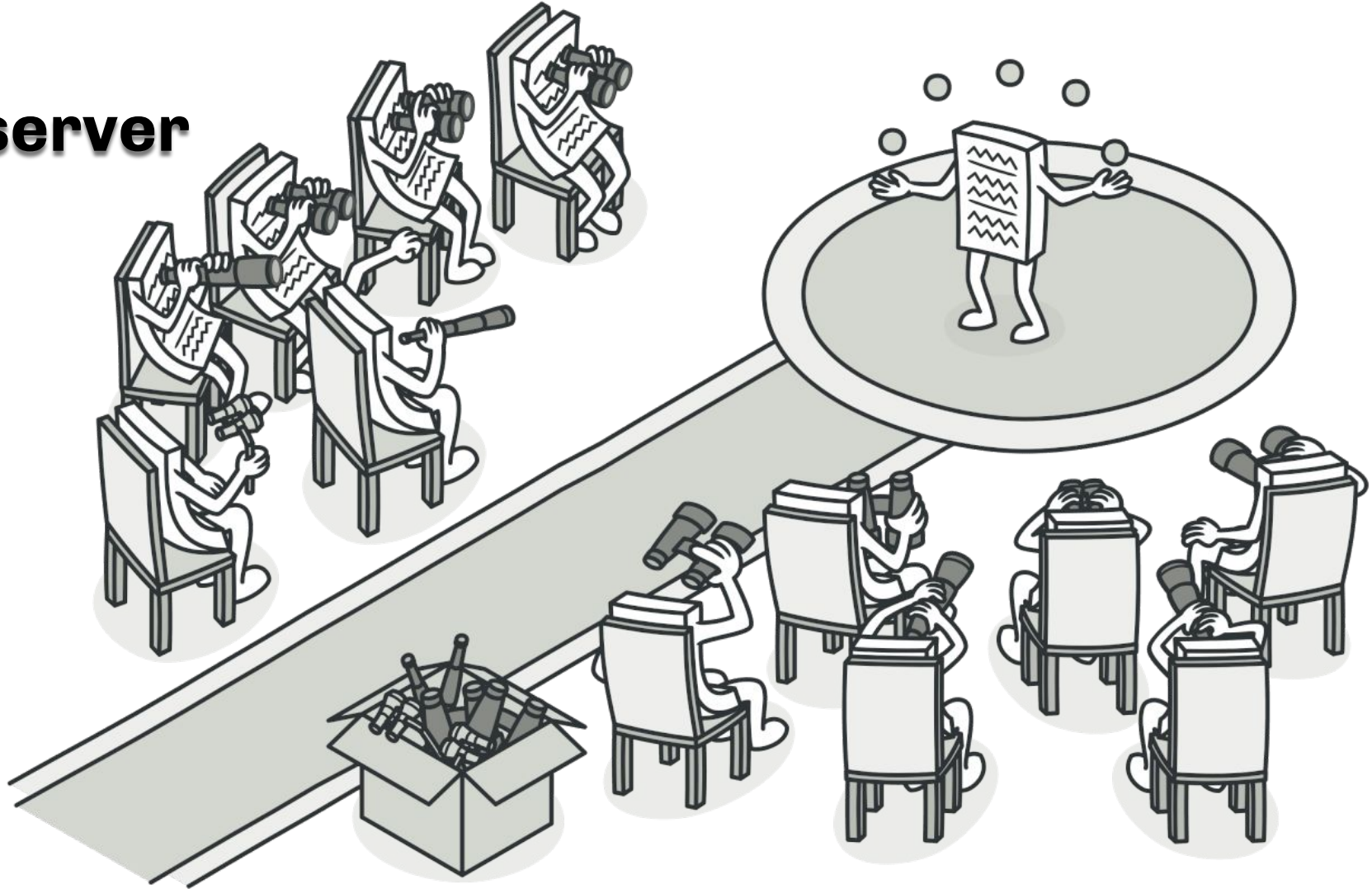
permite guardar y restaurar el estado interno de un objeto sin violar su encapsulamiento. Proporciona una forma de «capturar» el estado de un objeto en un momento dado y restaurarlo a ese estado más tarde.

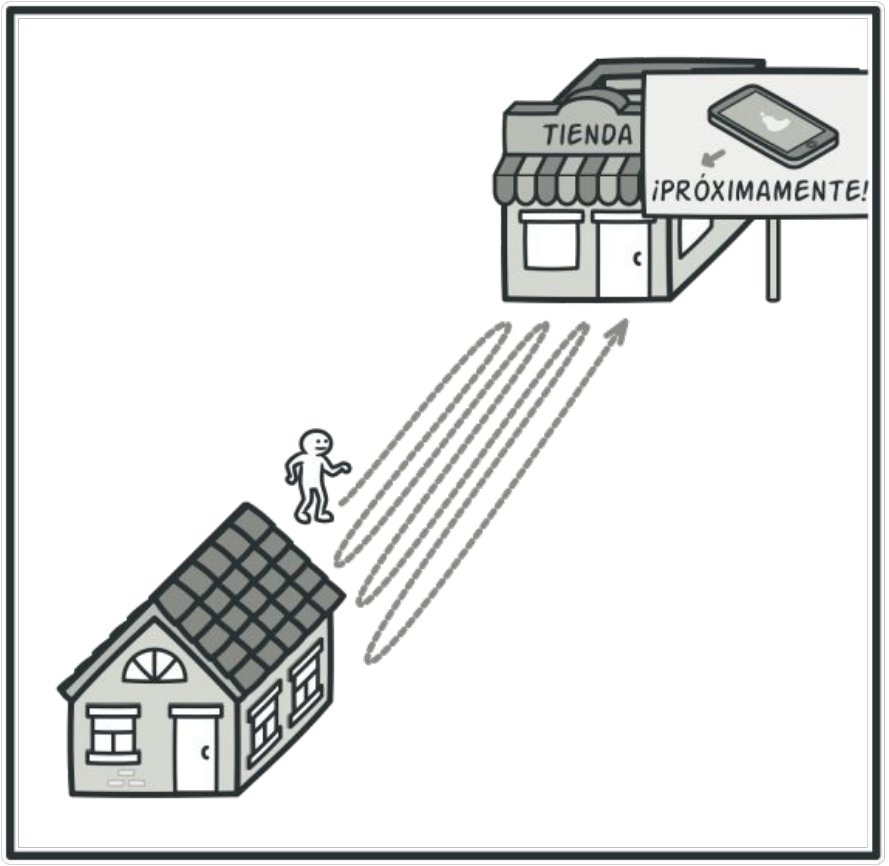

Caretaker
- history: List<Memento>
+ addMemento(memento: Memento): void
+ getMemento(index: int): Memento
No conoce el estado interno, solo lo pasa
Originator
- state: String
+ setState(state: String): void
+ getState(): String
+ createMemento(): Memento
+ restore(memento: Memento): void
Almacena
Crea y usa
Memento
// Puede ser una interfaz vacía o con un método para obtener una ID, fecha, etc.
// Interfaz restringida para Caretaker (no permite acceso a datos internos)
ConcreteMemento
- state: String
// Este método es solo accesible por Originator si Memento es una clase anidada
// o a través de un mecanismo de "friend" o package-private
+ ConcreteMemento(state: String)
+ getSavedState(): String

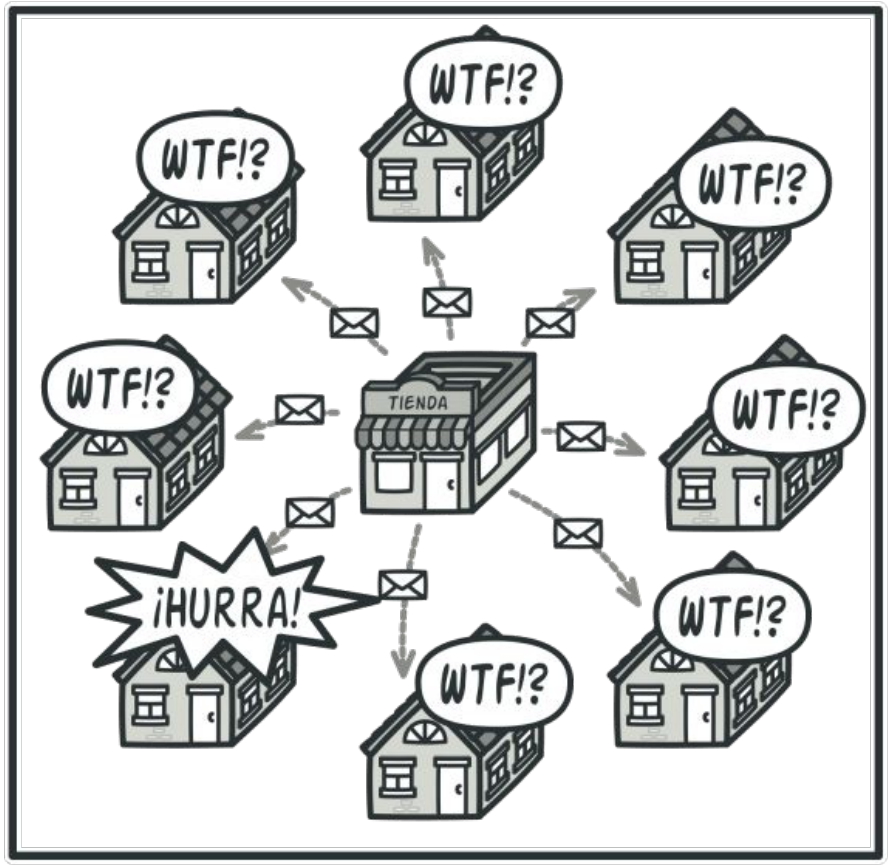

Client
Originator
Caretaker
Memento
1 new()
2 new()
3 setState("State 1")
4
5 createMemento()
6 «create»(state:"State 1")
7 m1
8 m1
9 addMemento(m1)
10
11 setState("State 2")
12
13 createMemento()
14 «create»(state:"State 2")
15 m2
16 m2
17 addMemento(m2)
18
19 getMemento(0)
20 mementoToRestore (m1)
21 restoreMemento(mementoToRestore)
22 getState()
23 "State 1"
24
25 getState()
26 "State 1"
Client
Originator
Caretaker
Memento


¿Preguntas?

Observer

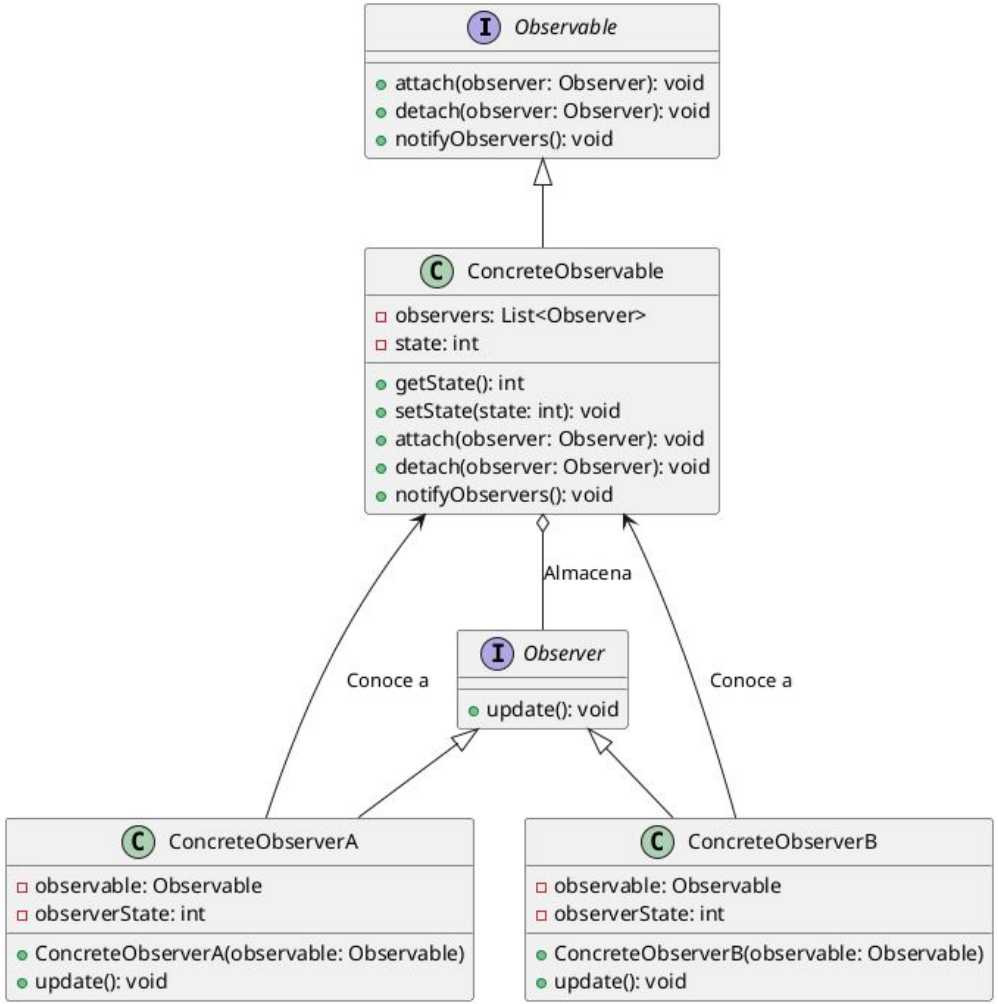



TIENDA
¡PRÓXIMAMENTE!


WTF!?
WTF!?
WTF!?
WTF!?
TIENDA
WTF!?
¡HURRA!
WTF!?
WTF!?

Observer

establece un mecanismo donde un objeto mantiene una lista de objetos interesados en su estado.
Cuando este cambia, les notifica automáticamente

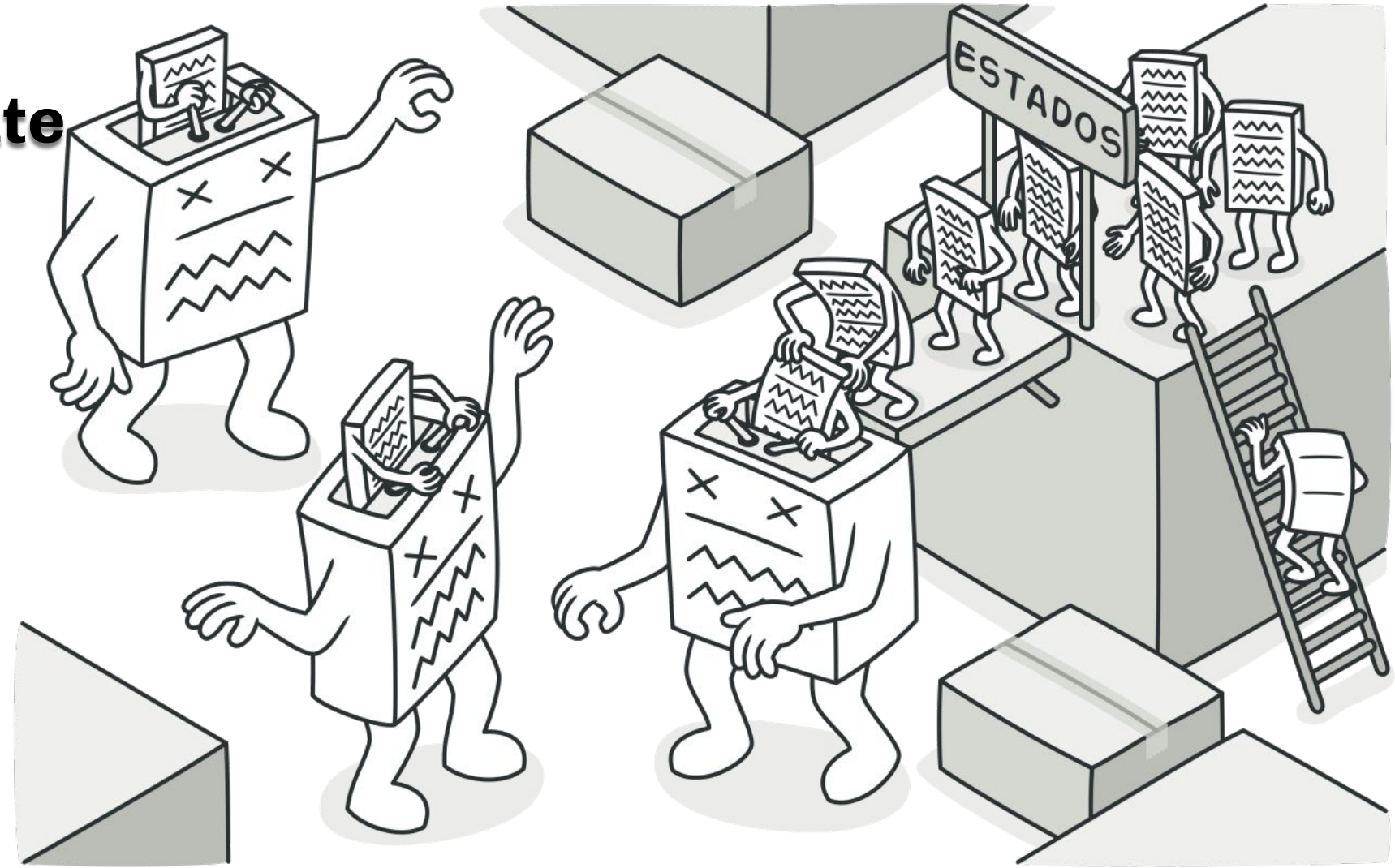

Observable
attach(observer: Observer): void
detach(observer: Observer): void
notifyObservers(): void
ConcreteObservable
observers: List<Observer>
state: int
getState(): int
setState(state: int): void
attach(observer: Observer): void
detach(observer: Observer): void
notifyObservers(): void
Almacena
Observer
update(): void
Conoce a
Conoce a
ConcreteObserverA
observable: Observable
observerState: int
ConcreteObserverA(observable: Observable)
update(): void
ConcreteObserverB
observable: Observable
observerState: int
ConcreteObserverB(observable: Observable)
update(): void

No nos llames, nosotros lo haremos

El principio Hollywood, inversión de Control

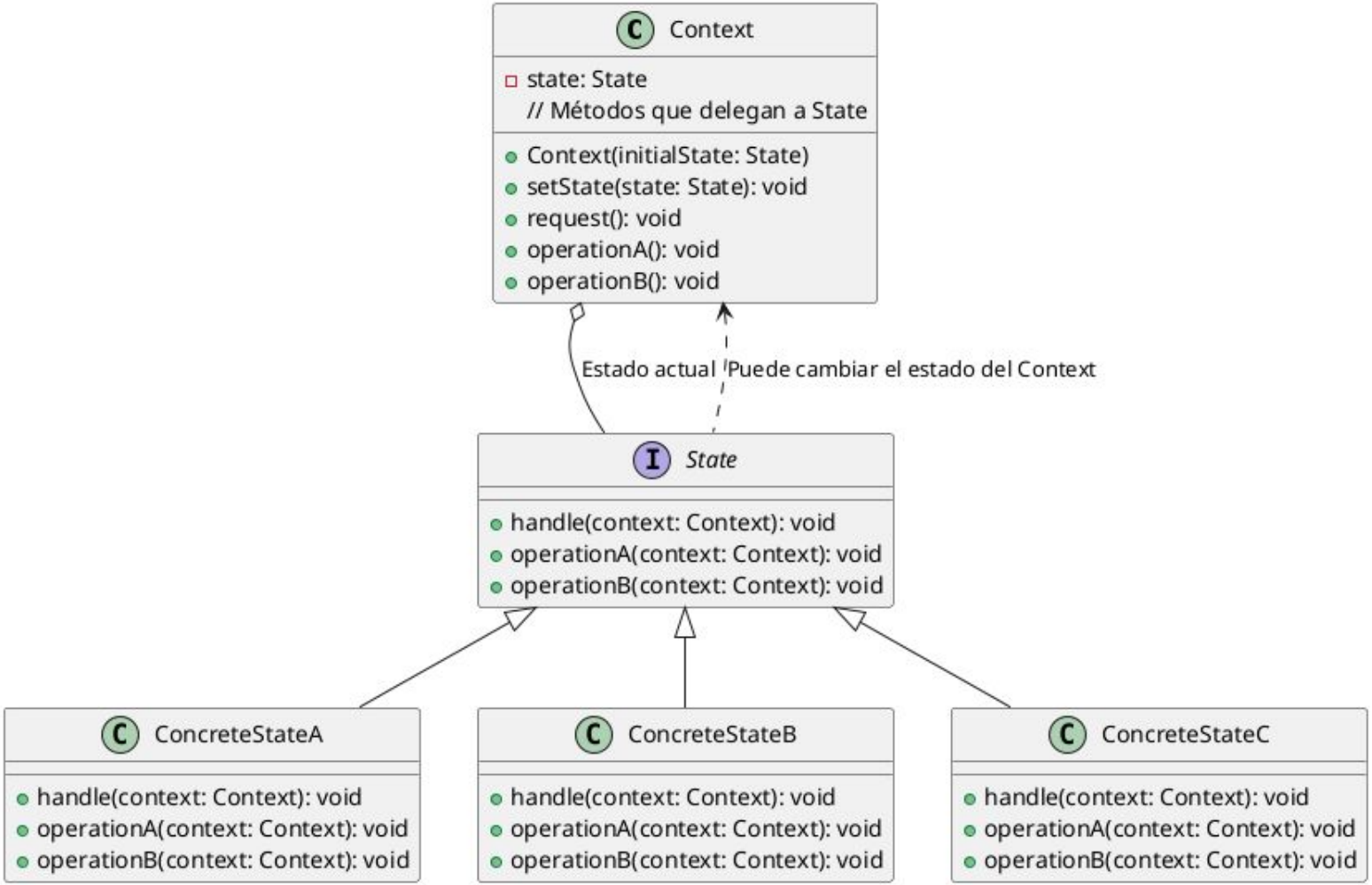

¿Preguntas?

State



State

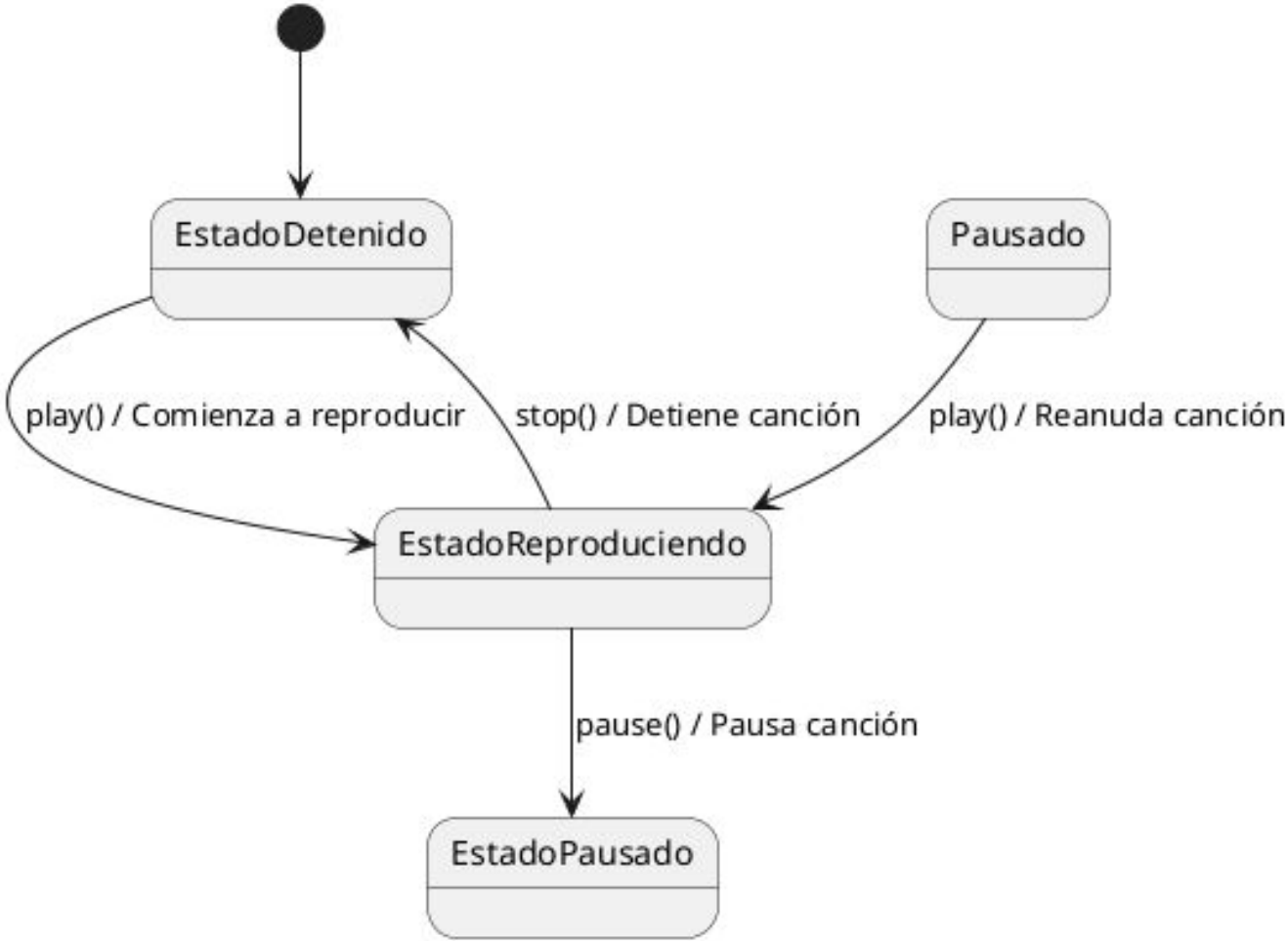
permite que un objeto altere su comportamiento cuando su estado interno cambia.
El objeto parece que cambia su clase.

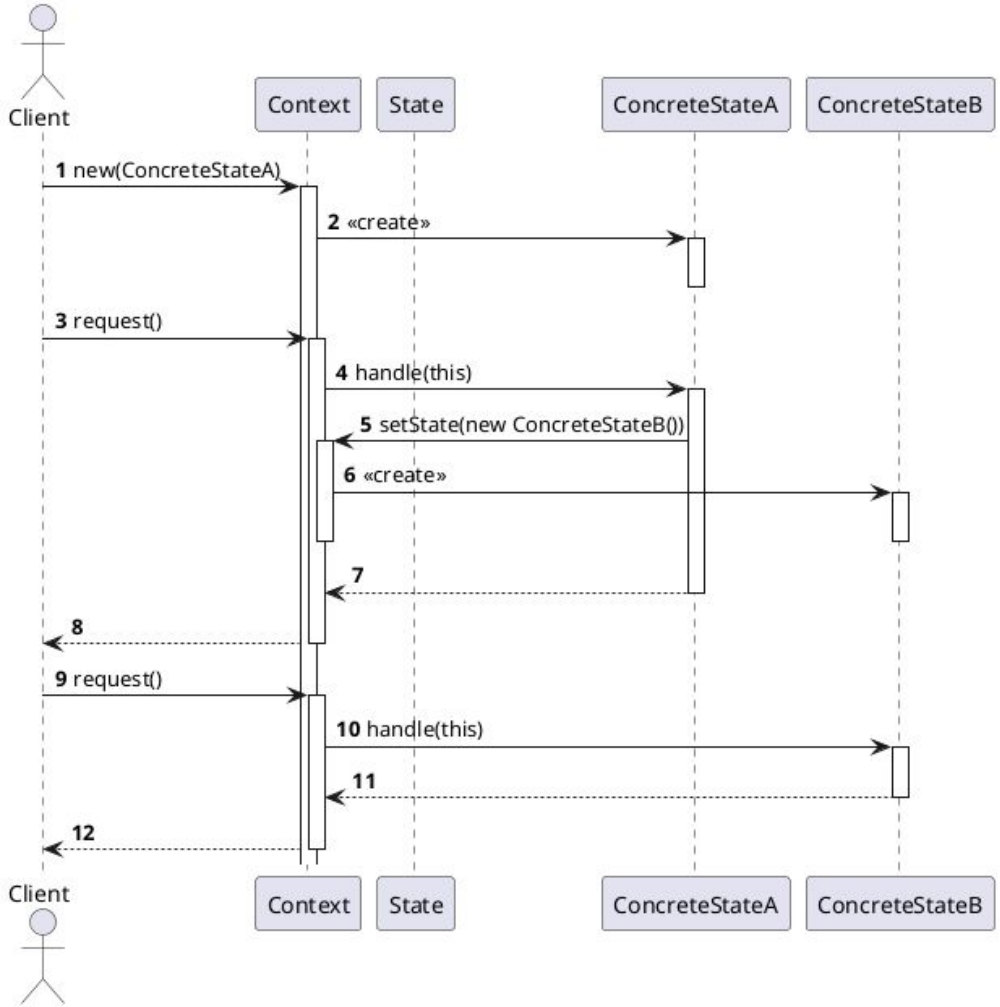

Context
state: State
// Métodos que delegan a State
Context(initialState: State)
setState(state: State): void
request(): void
operationA(): void
operationB(): void
Estado actual
Puede cambiar el estado del Context
State
handle(context: Context): void
operationA(context: Context): void
operationB(context: Context): void
ConcreteStateA
handle(context: Context): void
operationA(context: Context): void
operationB(context: Context): void
ConcreteStateB
handle(context: Context): void
operationA(context: Context): void
operationB(context: Context): void
ConcreteStateC
handle(context: Context): void
operationA(context: Context): void
operationB(context: Context): void

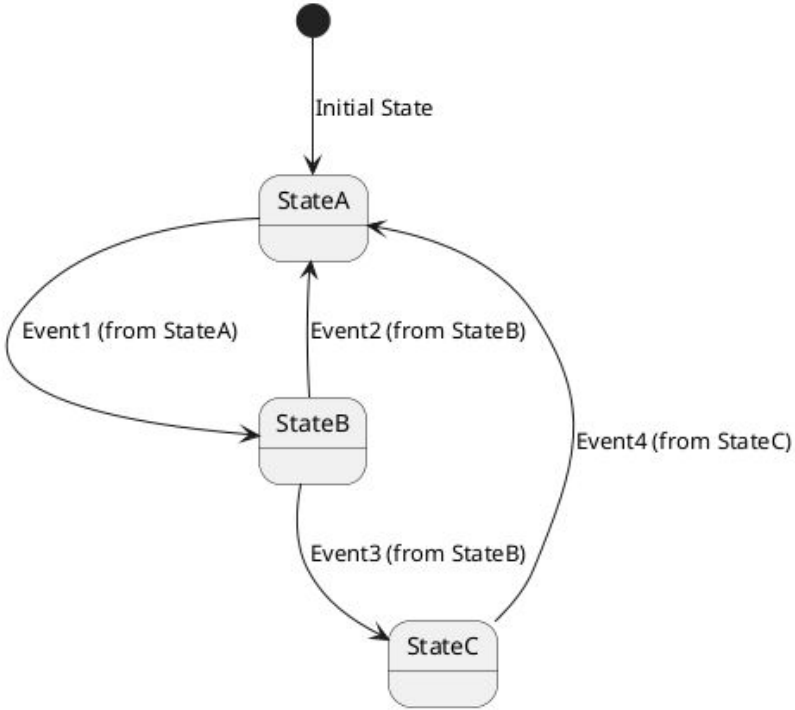
Máquina de estados

es un modelo de comportamiento que describe cómo un sistema pasa a través de un número finito de estados en respuesta a eventos o entradas.

Diagrama de Estados del Reproductor de Música




Client
Context
State
ConcreteStateA
ConcreteStateB
1 new(ConcreteStateA)
2 «create»
3 request()
4 handle(this)
5 setState(new ConcreteStateB())
6 «create»
7
8
9 request()
10 handle(this)
11
12
Client
Context
State
ConcreteStateA
ConcreteStateB

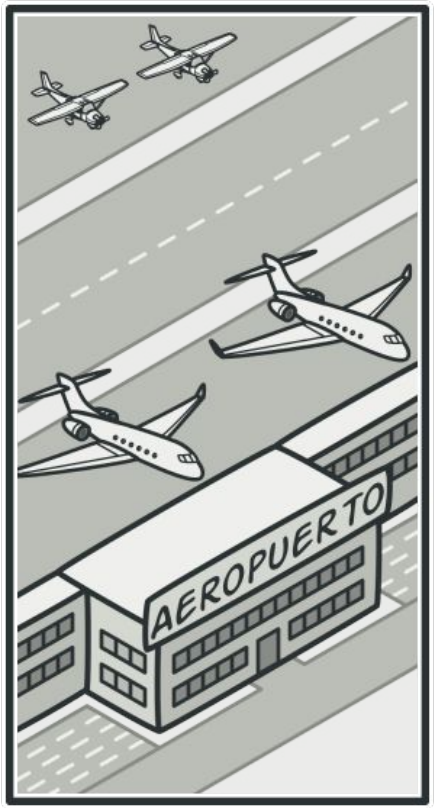

Initial State
StateA
Event1 (from StateA)
Event2 (from StateB)
StateB
Event4 (from StateC)
Event3 (from StateB)
StateC

{ No todas las acciones son validas en todo momento }

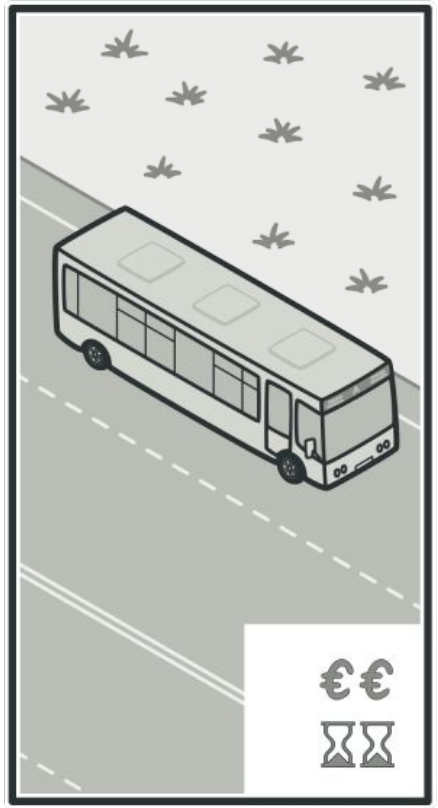
{ **Cada estado puede decidir *dinámicamente* como seguir** }


¿Preguntas?

Strategy


AEROPUERTO

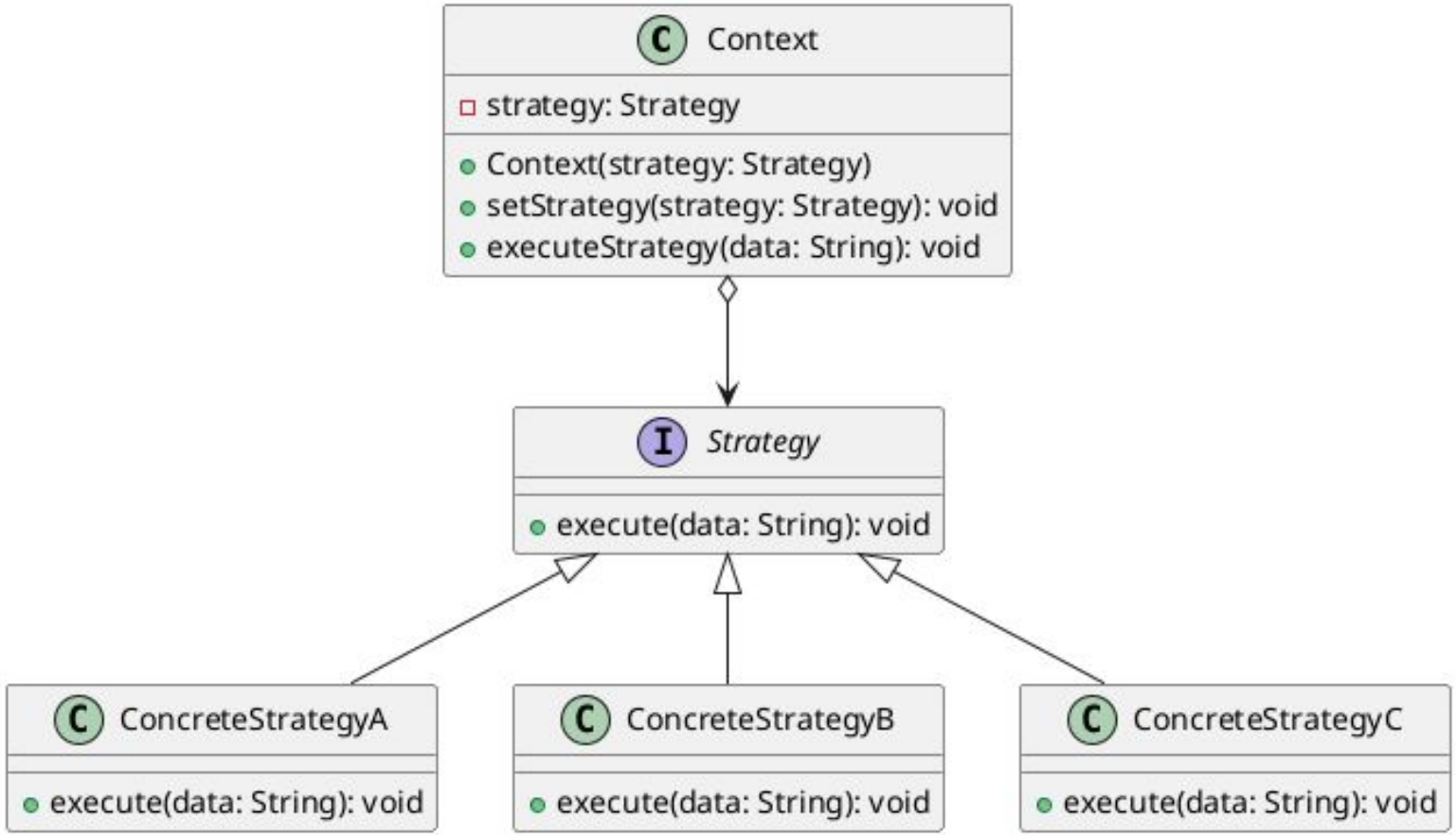

€0

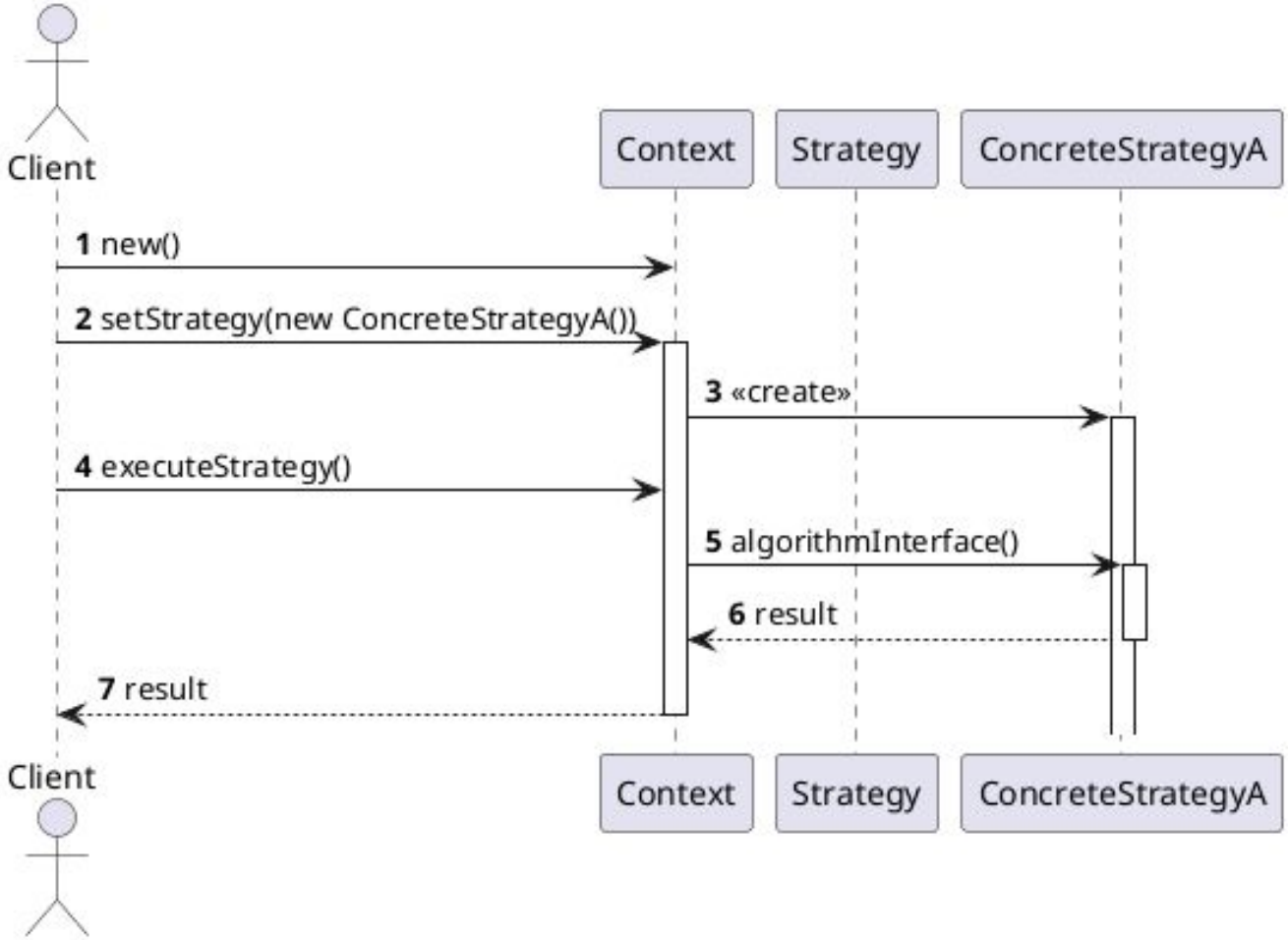

€€


€€€

Strategy

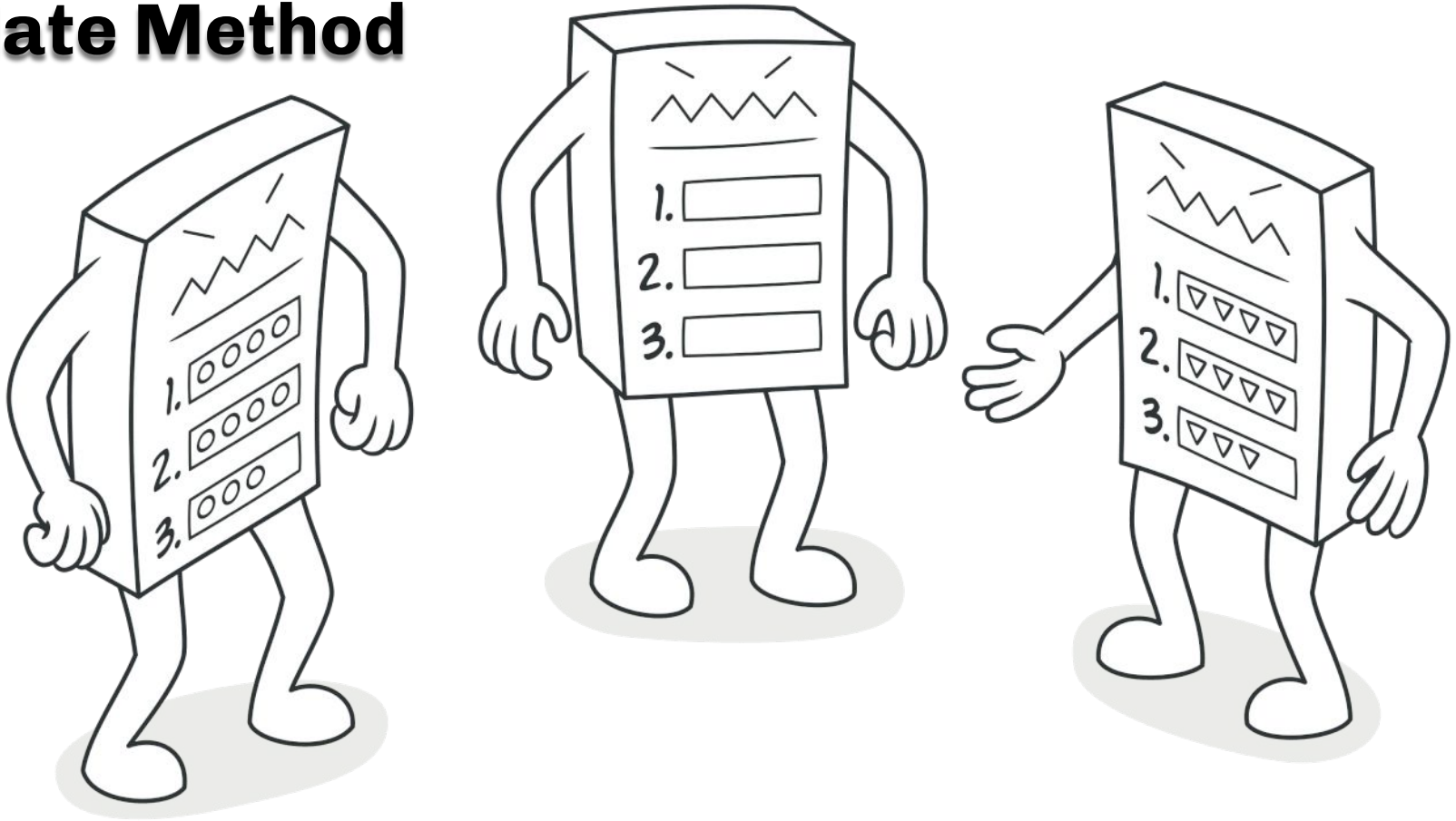
define una familia de algoritmos, los encapsula y los hace intercambiables.
El patrón permite que el algoritmo varíe independientemente de los clientes que lo utilizan.

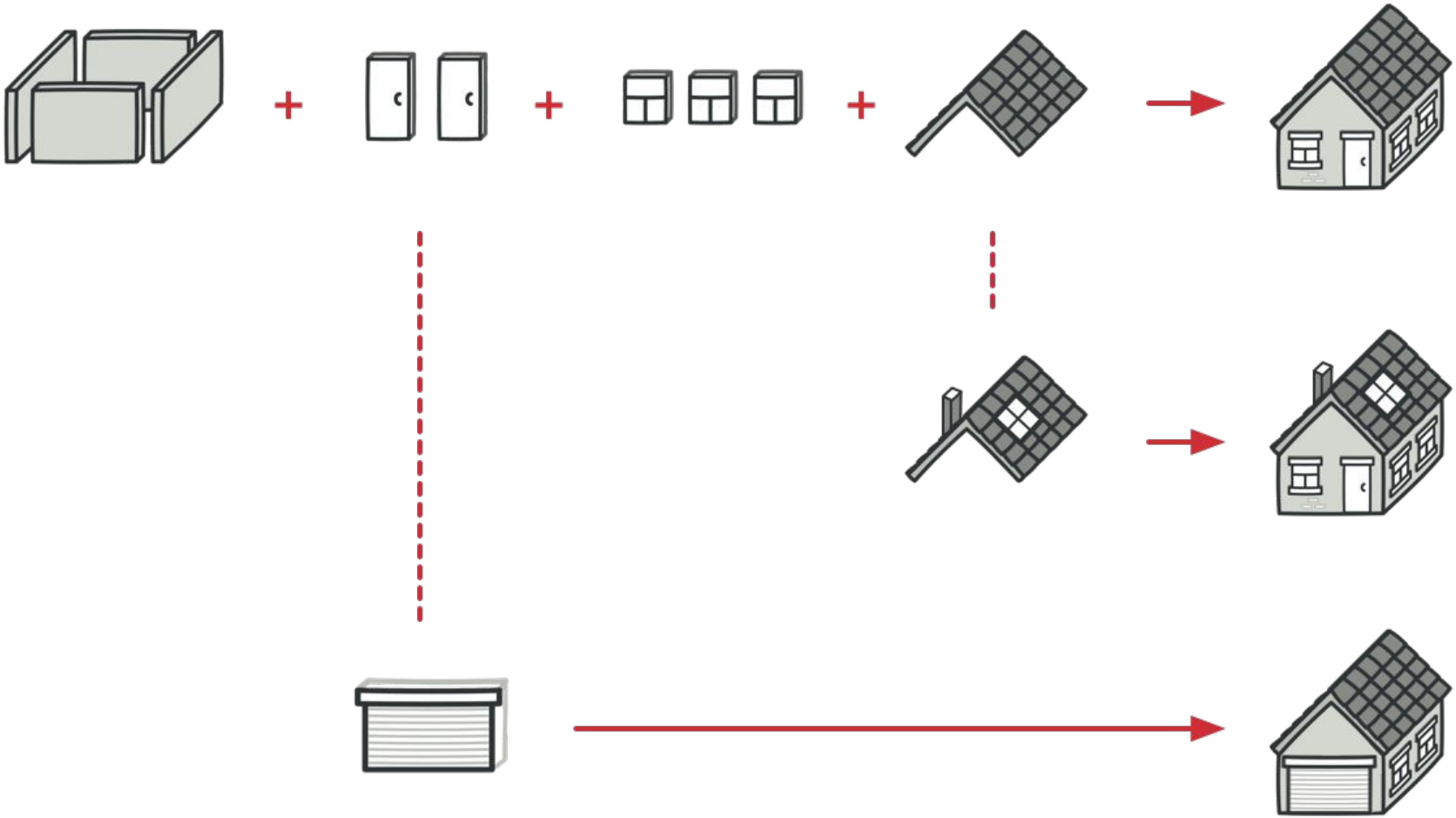

Context
strategy: Strategy
Context(strategy: Strategy)
setStrategy(strategy: Strategy): void
executeStrategy(data: String): void
Strategy
execute(data: String): void
ConcreteStrategyA
execute(data: String): void
ConcreteStrategyB
execute(data: String): void
ConcreteStrategyC
execute(data: String): void


Client
Context
Strategy
ConcreteStrategyA
1 new()
2 setStrategy(new ConcreteStrategyA())
3 «create»
4 executeStrategy()
5 algorithmInterface()
6 result
7 result
Client
Context
Strategy
ConcreteStrategyA


¿Preguntas?

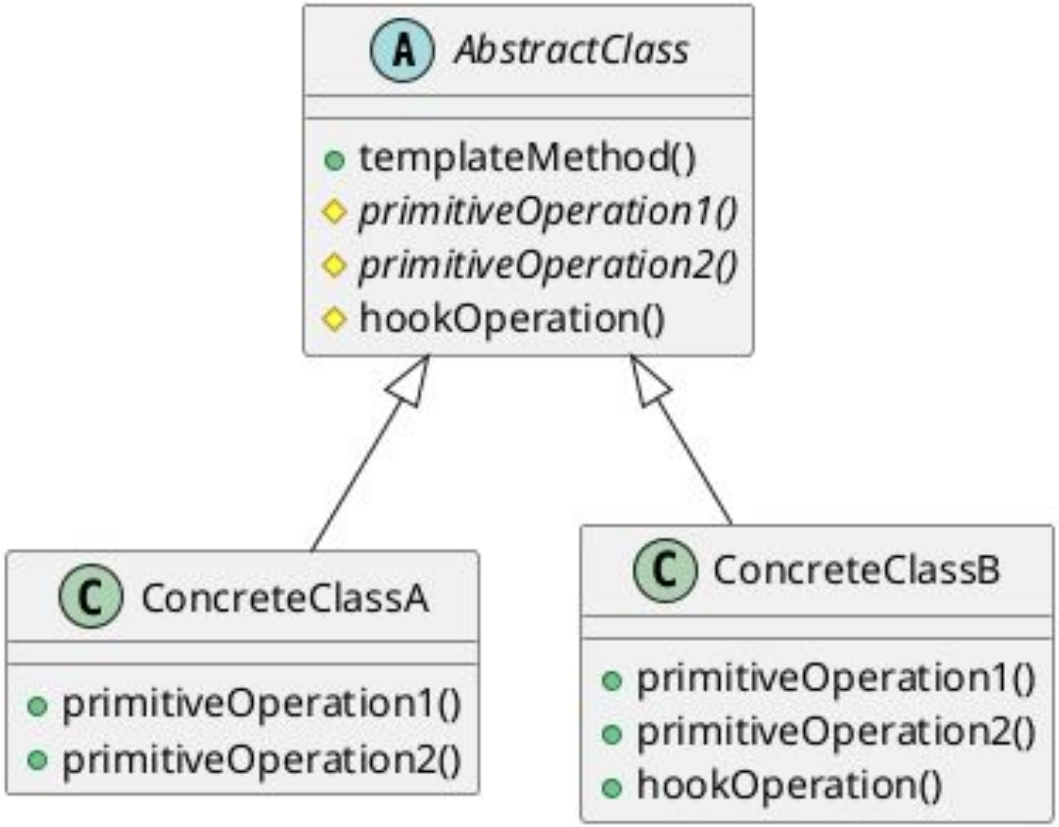
Template Method

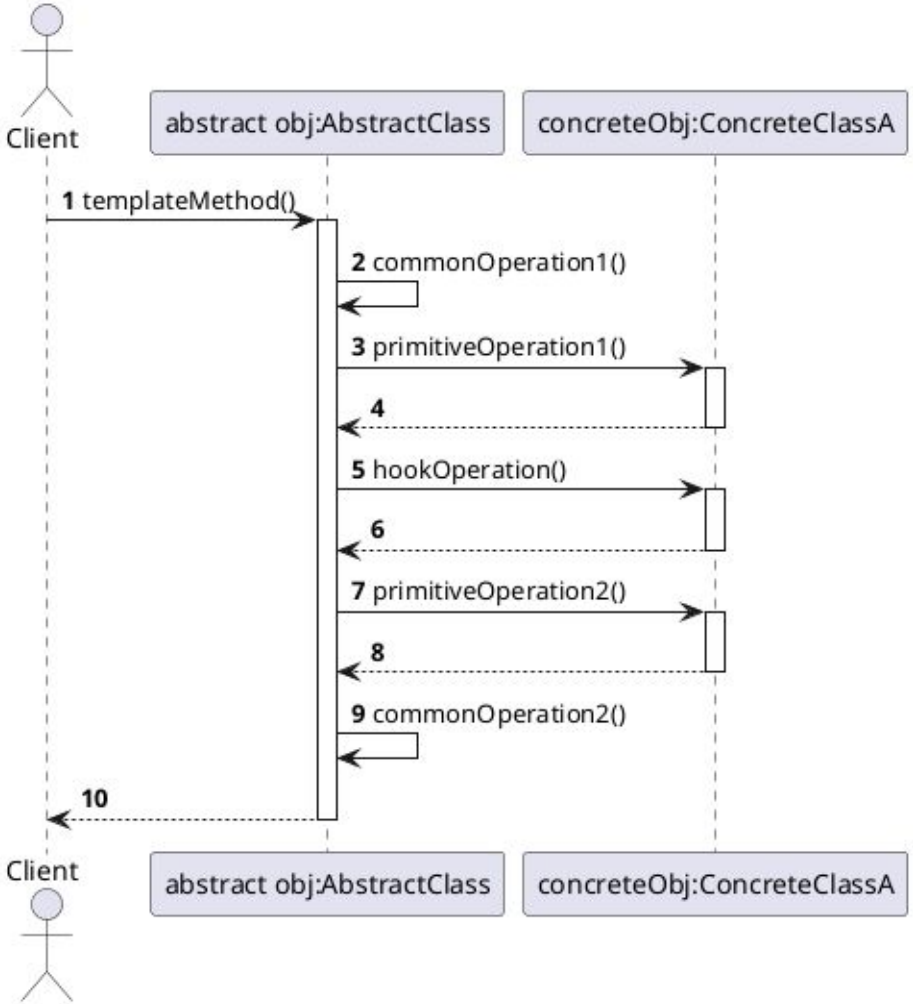




Template method

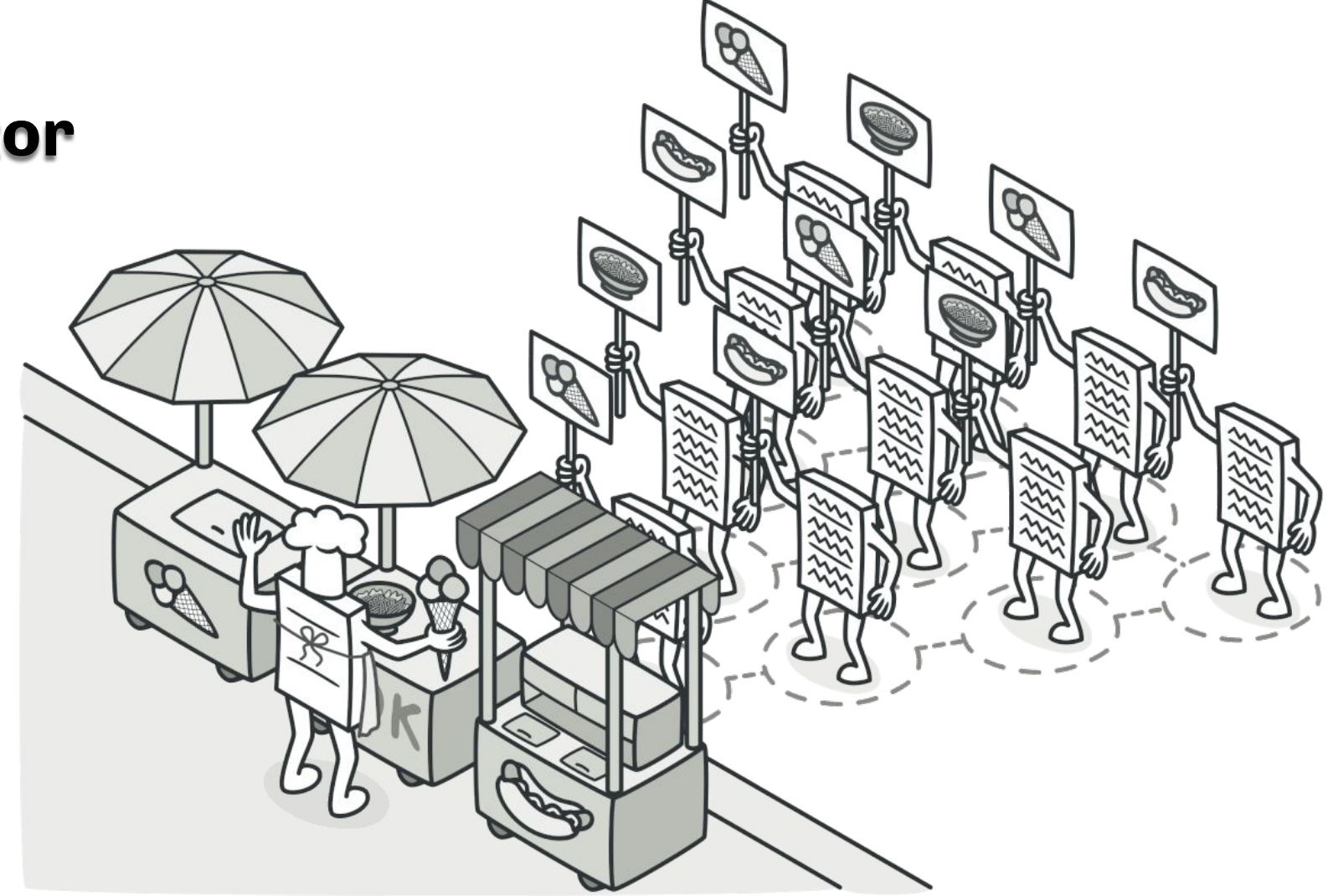
se centra en definir el marco de un algoritmo mientras permite que las subclases proporcionen las implementaciones específicas de ciertos pasos.

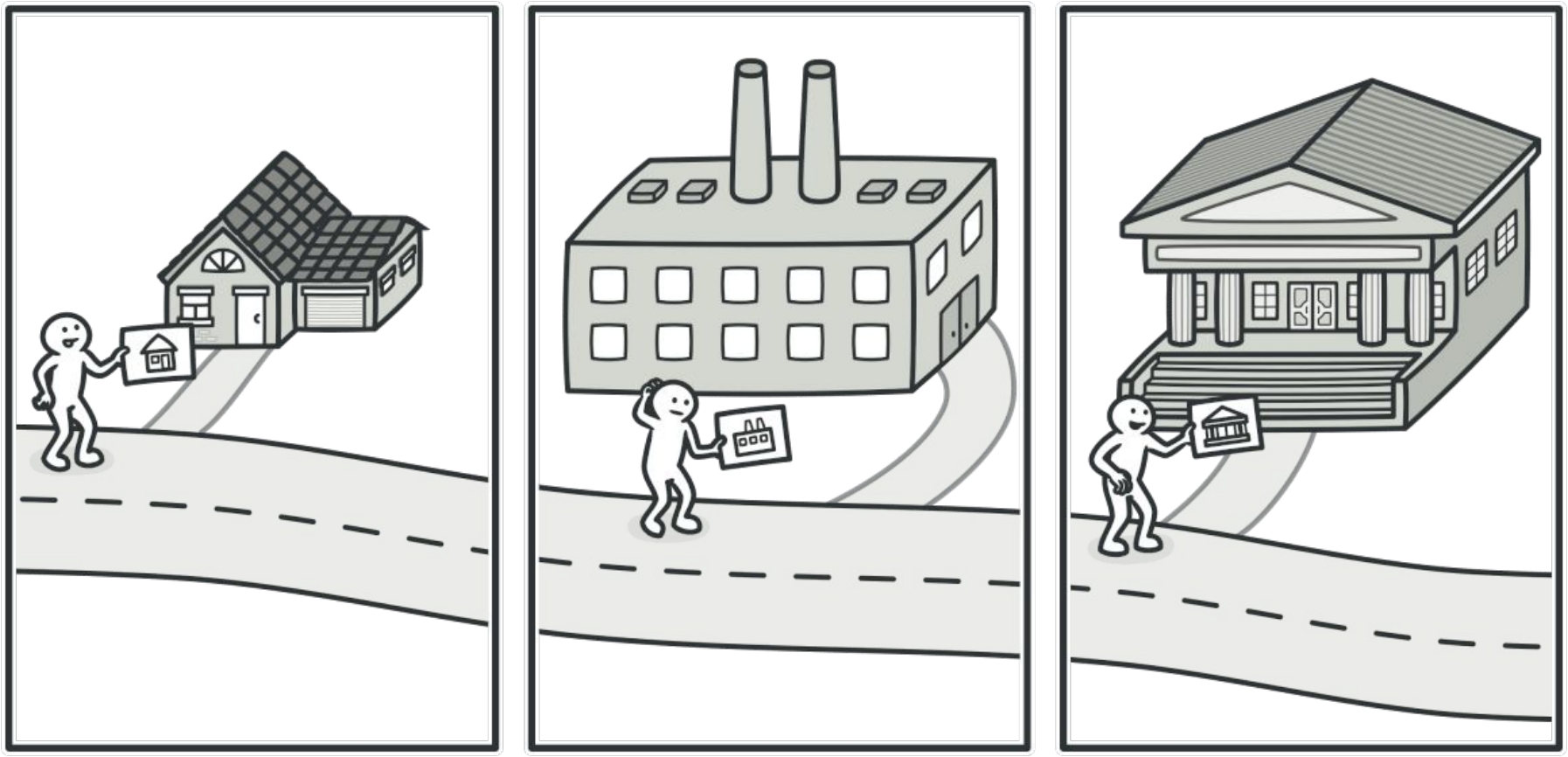

AbstractClass
templateMethod()
primitiveOperation1()
primitiveOperation2()
hookOperation()
ConcreteClassA
primitiveOperation1()
primitiveOperation2()
ConcreteClassB
primitiveOperation1()
primitiveOperation2()
hookOperation()


Client
abstract obj:AbstractClass
concreteObj:ConcreteClassA
1 templateMethod()
2 commonOperation1()
3 primitiveOperation1()
4
5 hookOperation()
6
7 primitiveOperation2()
8
9 commonOperation2()
10
Client
abstract obj:AbstractClass
concreteObj:ConcreteClassA


¿Preguntas?

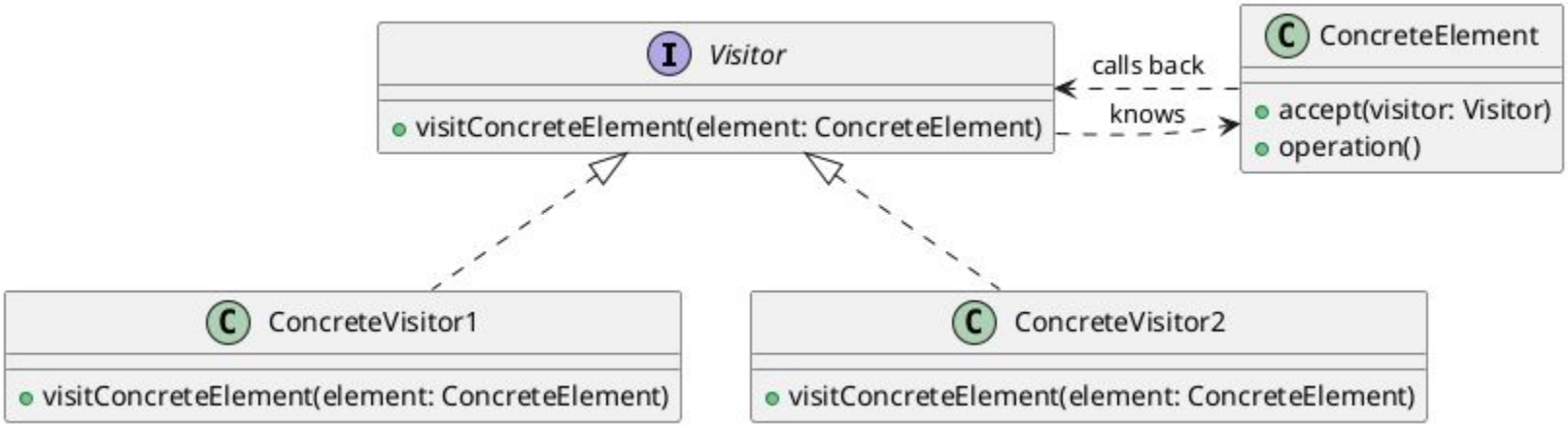
Visitor

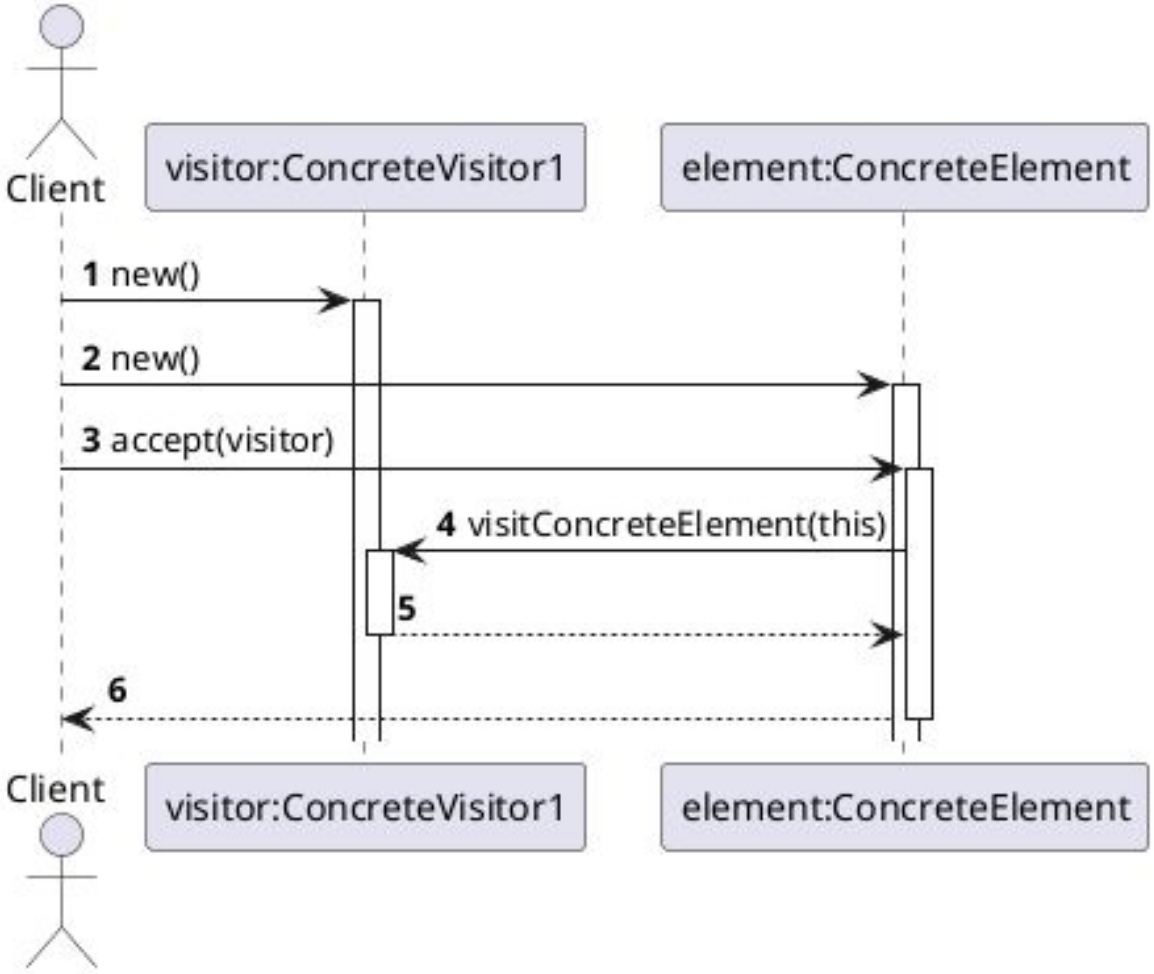


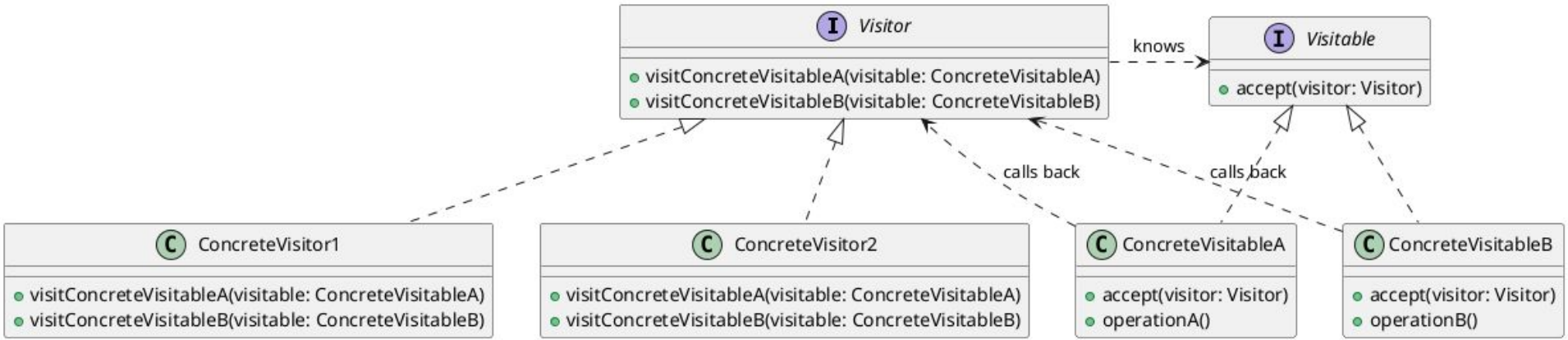


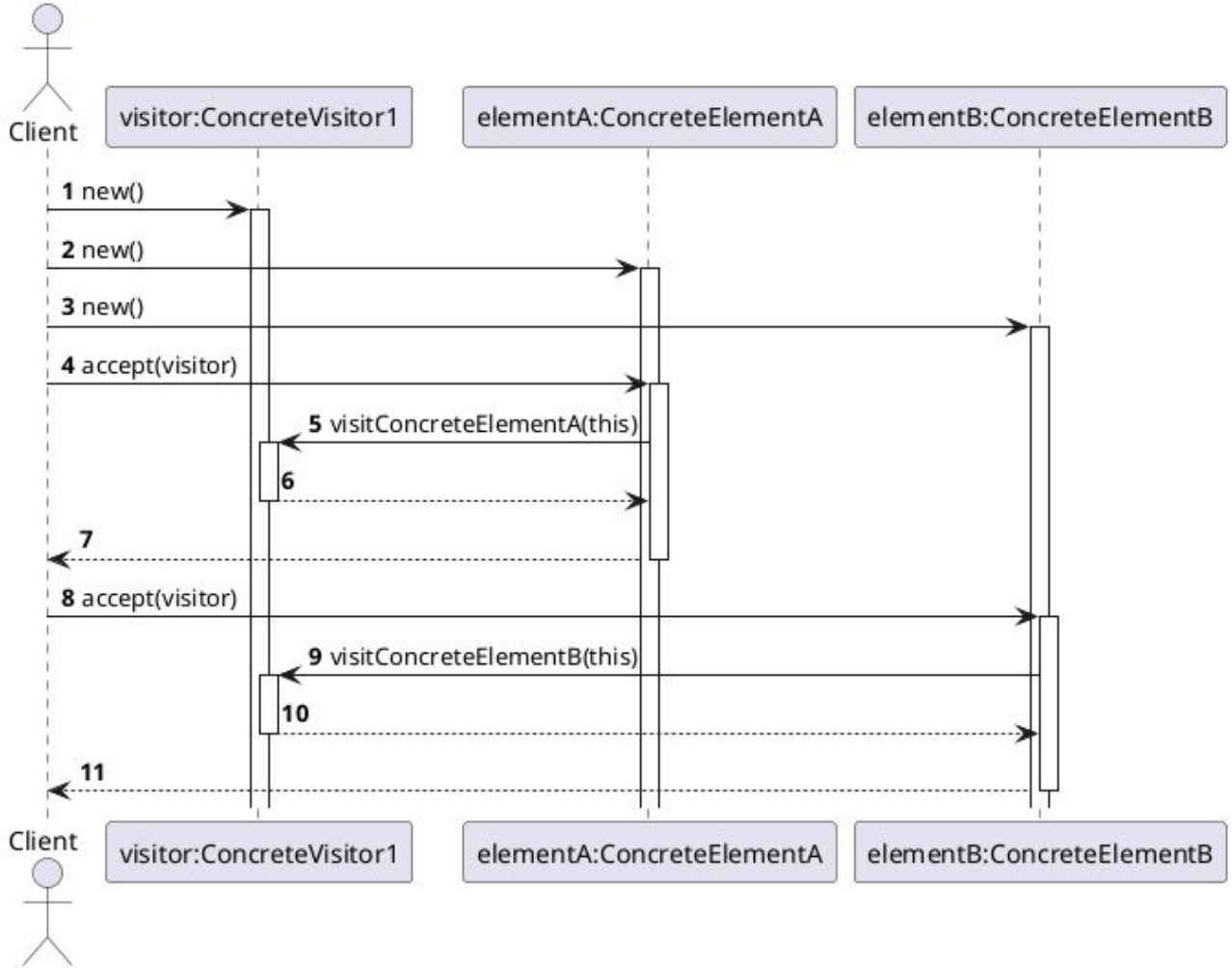
Visitor

permite separar un algoritmo de la estructura de objetos sobre la que opera.


Visitor
visitConcreteElement(element: ConcreteElement)
calls back
knows
ConcreteElement
accept(visitor: Visitor)
operation()
ConcreteVisitor1
visitConcreteElement(element: ConcreteElement)
ConcreteVisitor2
visitConcreteElement(element: ConcreteElement)


Client
visitor:ConcreteVisitor1
element:ConcreteElement
1 new()
2 new()
3 accept(visitor)
4 visitConcreteElement(this)
5
6
Client
visitor:ConcreteVisitor1
element:ConcreteElement


Visitor
visitConcreteVisitableA(visitable: ConcreteVisitableA)
visitConcreteVisitableB(visitable: ConcreteVisitableB)
knows
Visitable
accept(visitor: Visitor)
calls back
calls back
ConcreteVisitor1
visitConcreteVisitableA(visitable: ConcreteVisitableA)
visitConcreteVisitableB(visitable: ConcreteVisitableB)
ConcreteVisitor2
visitConcreteVisitableA(visitable: ConcreteVisitableA)
visitConcreteVisitableB(visitable: ConcreteVisitableB)
ConcreteVisitableA
accept(visitor: Visitor)
operationA()
ConcreteVisitableB
accept(visitor: Visitor)
operationB()


Client
visitor:ConcreteVisitor1
elementA:ConcreteElementA
elementB:ConcreteElementB
1 new()
2 new()
3 new()
4 accept(visitor)
5 visitConcreteElementA(this)
6
7
8 accept(visitor)
9 visitConcreteElementB(this)
10
11
Client
visitor:ConcreteVisitor1
elementA:ConcreteElementA
elementB:ConcreteElementB

{ Combinado con Iterator, la forma de recorrer y que hacer es completamente dinámico }

Otros conceptos

`null` Object

En lugar de usar `null`, se usa un objeto que implementa la misma interfaz, pero con comportamiento vacío o desactivado.

Instanciación peresoza

Retrasa la creación de un objeto hasta el momento en que se necesita.

Objeto parcial

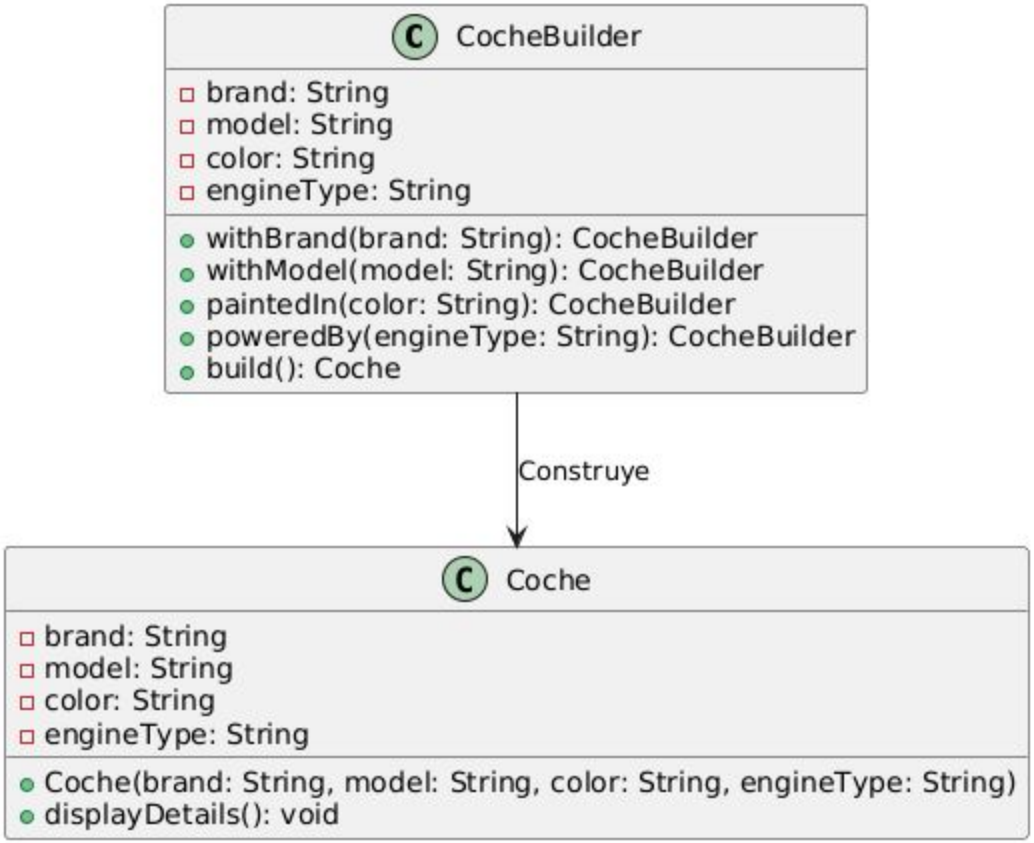
Se utiliza cuando solo se necesita una parte de un objeto grande, útil para optimizar rendimiento.

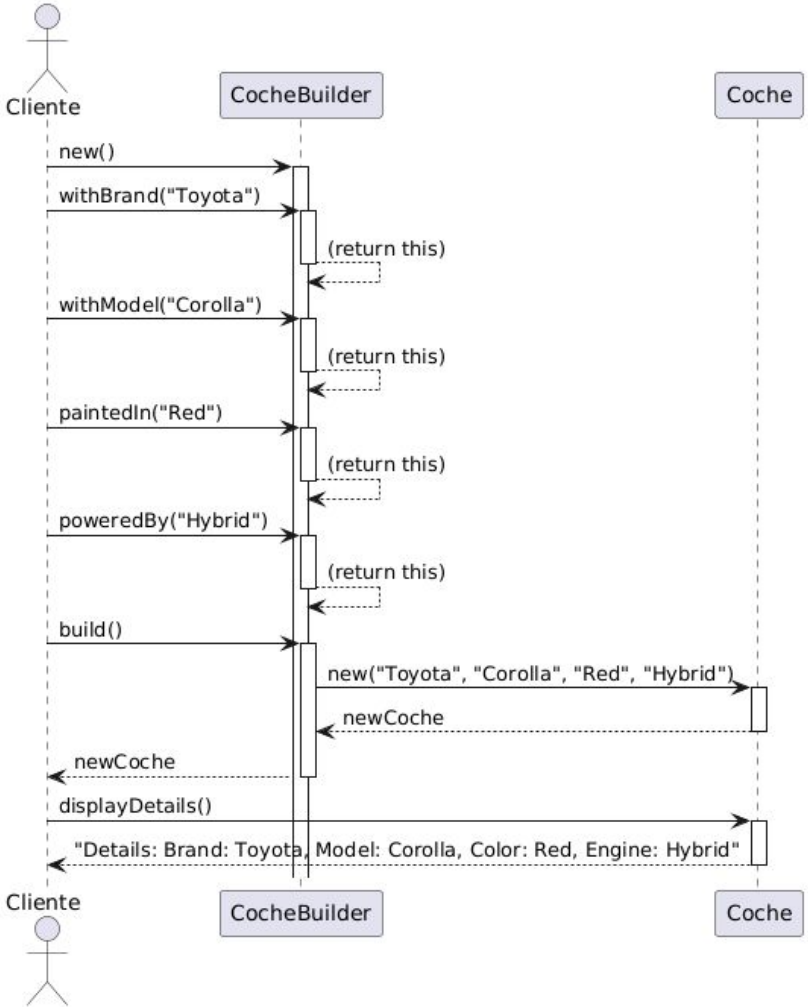
Objeto de consulta

Se crea una instancia de una clase de negocio de forma incompleta para ser utilizada como argumento.

Fluent Interface

Utilizado en Builder, simplifica la creación de objetos complejos, apilando llamadas a métodos.


CocheBuilder
brand: String
model: String
color: String
engineType: String
withBrand(brand: String): CocheBuilder
withModel(model: String): CocheBuilder
paintedIn(color: String): CocheBuilder
poweredBy(engineType: String): CocheBuilder
build(): Coche
Construye
Coche
brand: String
model: String
color: String
engineType: String
Coche(brand: String, model: String, color: String, engineType: String)
displayDetails(): void


Cliente
CocheBuilder
Coche
new()
withBrand("Toyota")
(return this)
withModel("Corolla")
(return this)
paintedIn("Red")
(return this)
poweredBy("Hybrid")
(return this)
build()
new("Toyota", "Corolla", "Red", "Hybrid")
newCoche
newCoche
displayDetails()
"Details: Brand: Toyota, Model: Corolla, Color: Red, Engine: Hybrid"
Cliente
CocheBuilder
Coche

TP10

Arrays y patrones


¿Preguntas?