Patrones de diseño III

XV
III
2025

UNRN
Universidad Nacional
de **Río Negro**

r27

equals / hashCode + herencia

Dada una Persona

```
class Persona {
private String nombre;
private int edad;

@Override
public int hashCode() {
    return Objects.hash(nombre, edad);
}
```

```
@Override
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null) {
        return false;
    }
    if (o instanceof Persona persona) {
        return edad == persona.edad &&
                Objects.equals(nombre, persona.nombre);
    } else {
        return false;
    }
}
```

Podemos definir un Empleado:

```
class Empleado extends Persona {
private String legajo;
private long salario;

@Override
public int hashCode() {
  return Objects.hash(
    super.hashCode(), legajo, salario);
}
```

```
@Override
public boolean equals(Object otro) {
  if (this == otro) {
    return true;
  }
  if (otro instanceof Empleado empleado) {
    return super.equals(empleado) &&
       Objects.equals(legajo, empleado.legajo)
&&
       (salario == empleado.salario);
  } else {
    return false;
  }
}
```

Para reutilizar lo que ya está *arriba*

Estructurales

Estructural

S

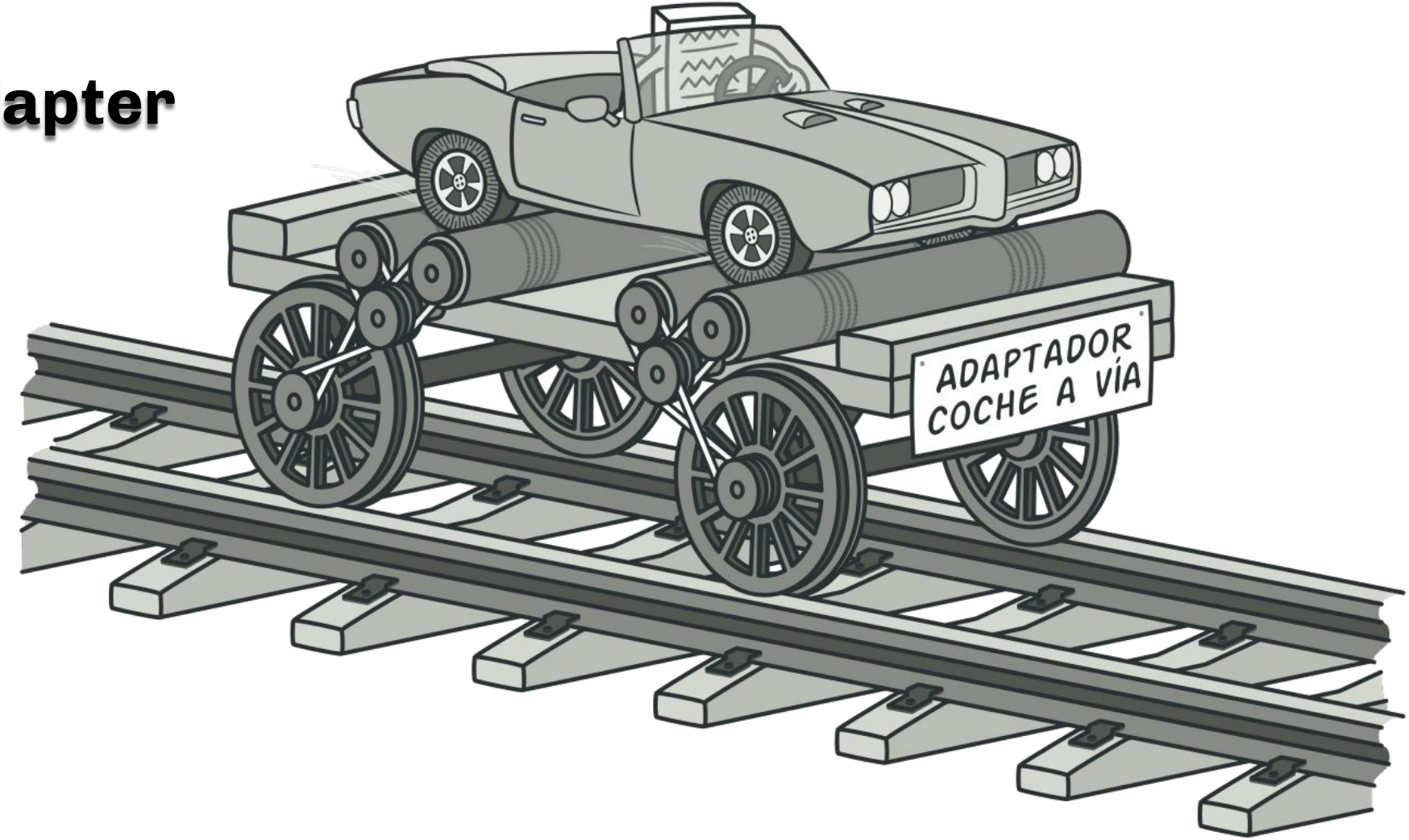
Se enfocan en cómo componer clases y objetos para formar estructuras más grandes y flexibles.

Estructurales

- Adapter
- Bridge
- Composite
- Decorator

- Facade
- Flyweight
- Proxy

Adapter



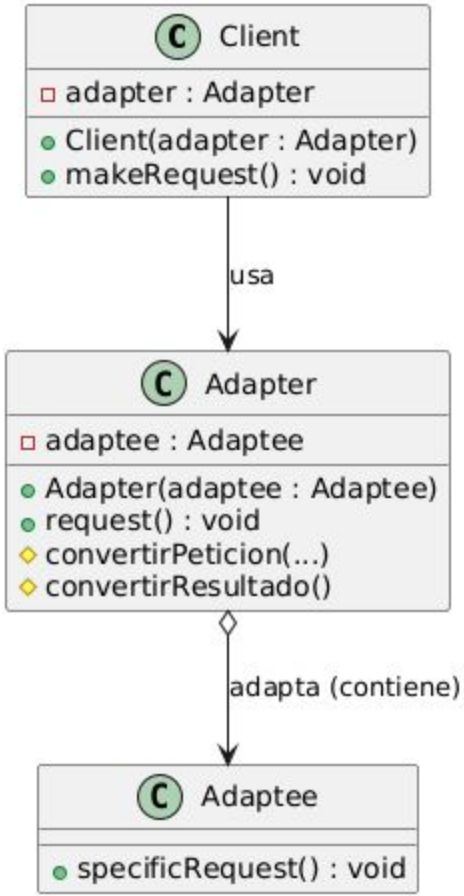
Adapter

Convierte la interfaz de una clase en otra interfaz que el cliente espera, permitiendo que clases con interfaces incompatibles trabajen juntas

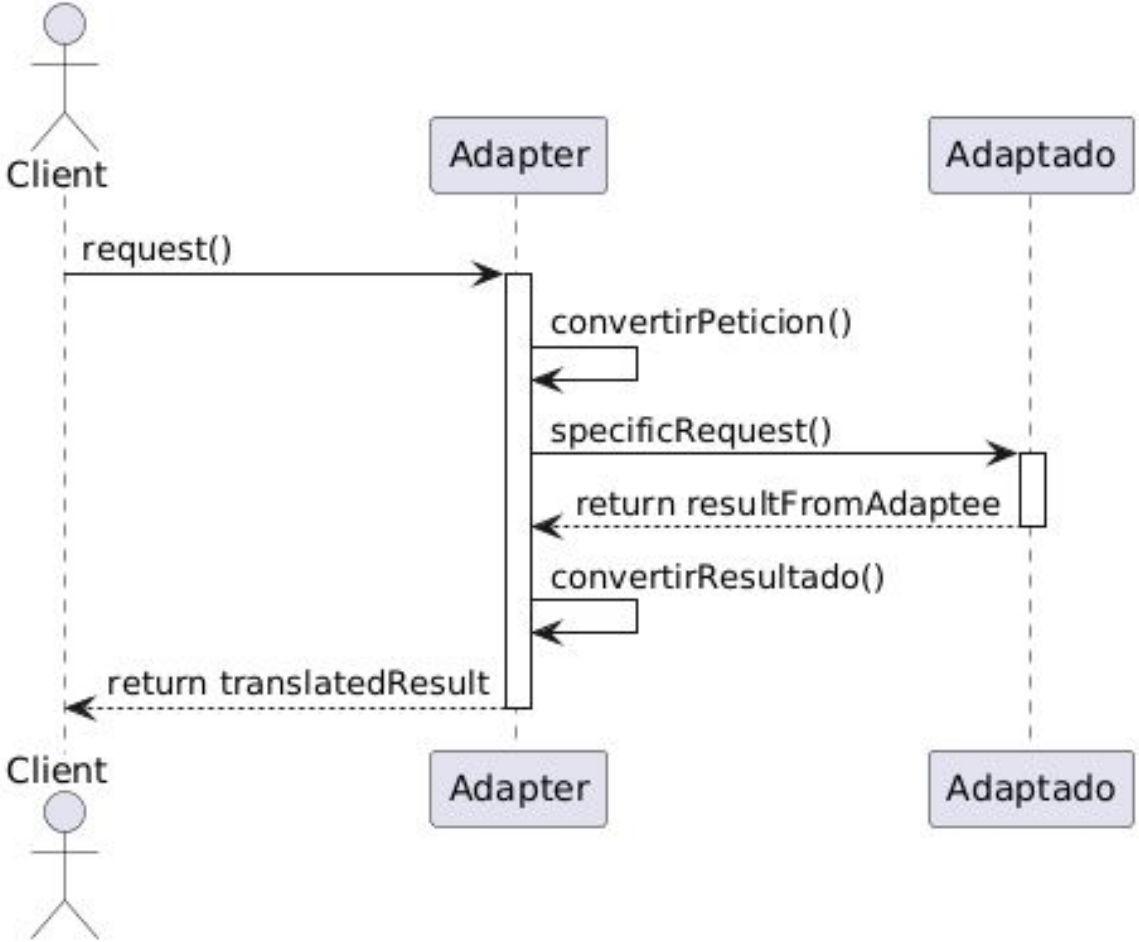
VIAJAR AL EXTRANJERO

1ª VEZ

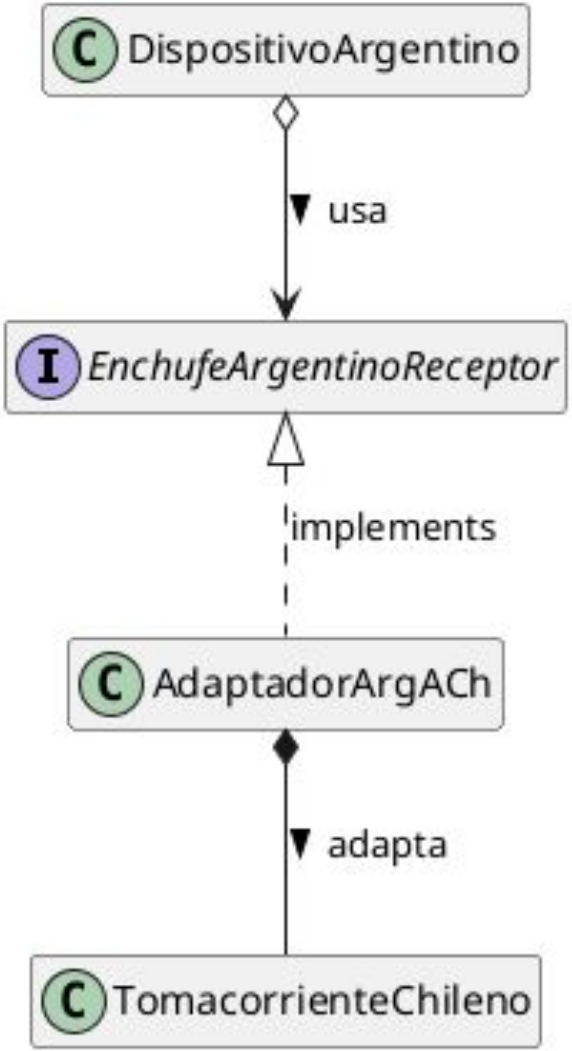
10ª VEZ

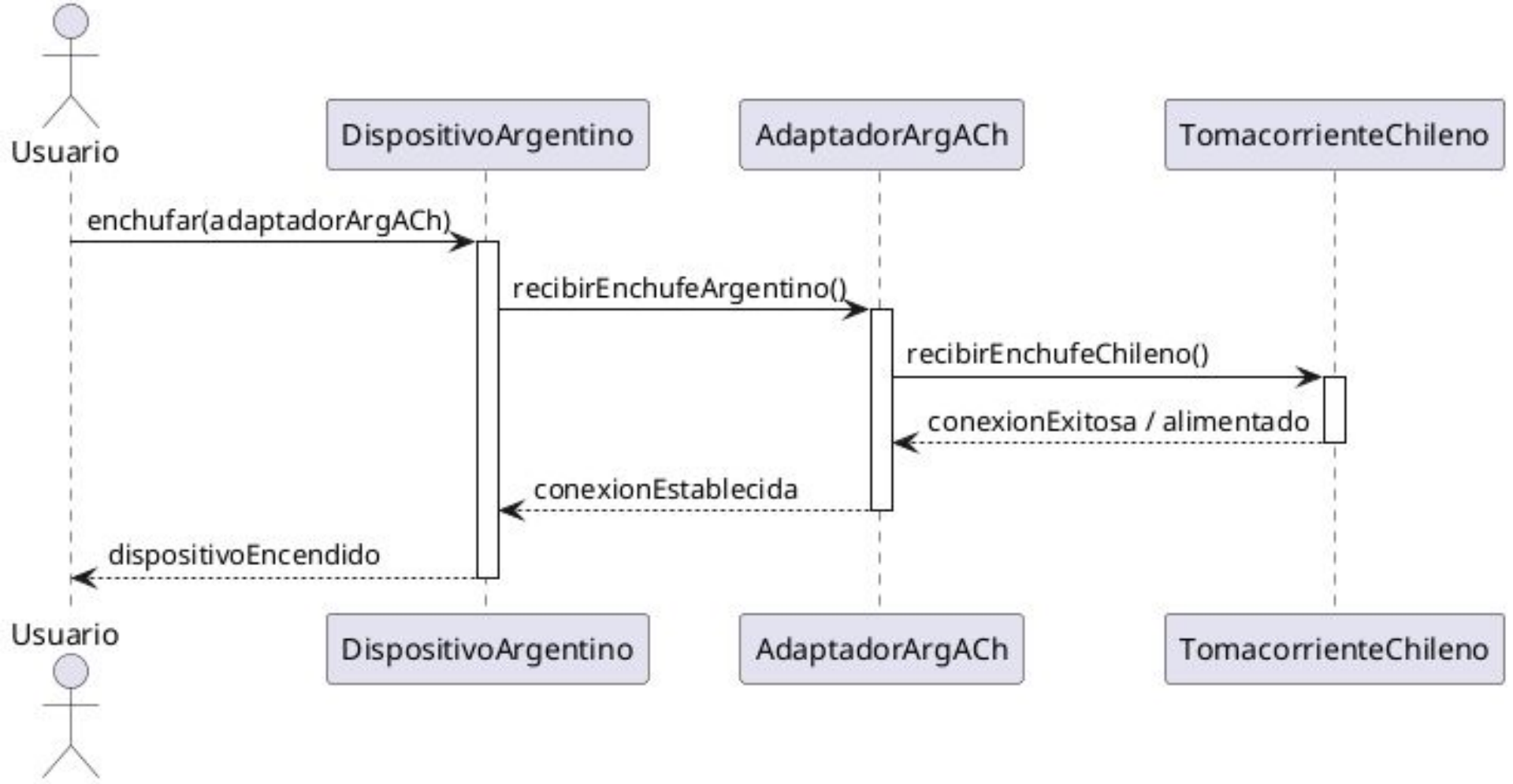

Client
adapter : Adapter
Client(adapter : Adapter)
makeRequest() : void
usa
Adapter
adaptee : Adaptee
Adapter(adaptee : Adaptee)
request() : void
convertirPeticion(...)
convertirResultado()
adapta (contiene)
Adaptee
specificRequest() : void

Puede ir con una interfaz


Client
Adapter
Adaptado
request()
convertirPeticion()
specificRequest()
return resultFromAdaptee
convertirResultado()
return translatedResult
Client
Adapter
Adaptado

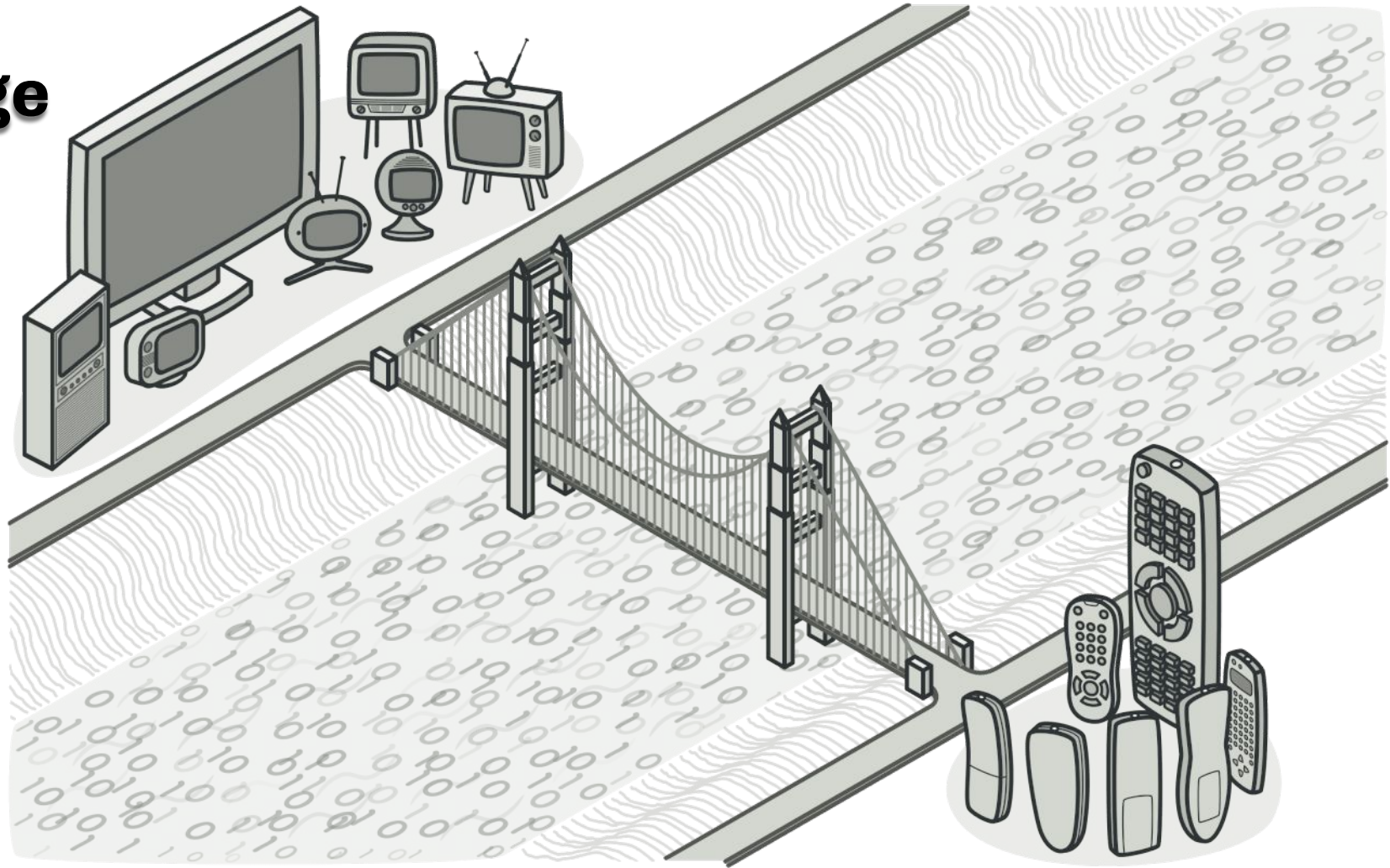

TRAVEL
UNIVERSAL
ADAPTOR
SURGE
PROTECTOR
CITIBANK


DispositivoArgentino
usa
EnchufeArgentinoReceptor
implements
AdaptadorArgACh
adapta
TomacorrienteChileno


Usuario
DispositivoArgentino
AdaptadorArgACh
TomacorrienteChileno
enchufar(adaptadorArgACh)
recibirEnchufeArgentino()
recibirEnchufeChileno()
conexionExitosa / alimentado
conexionEstablecida
dispositivoEncendido
Usuario
DispositivoArgentino
AdaptadorArgACh
TomacorrienteChileno

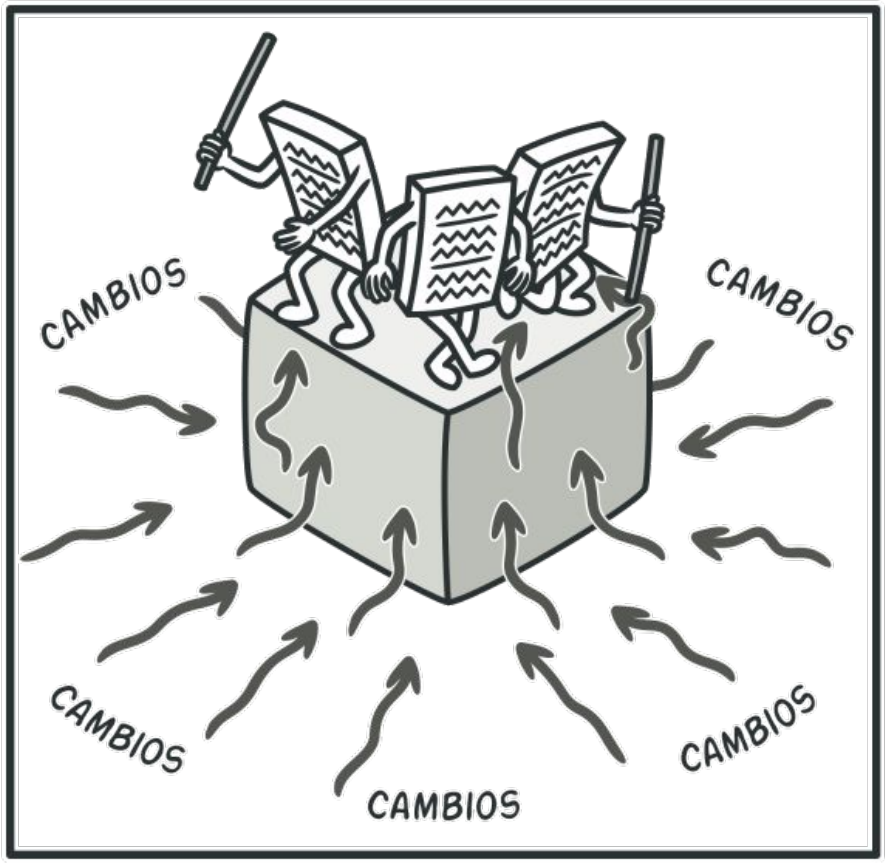

¿Preguntas?

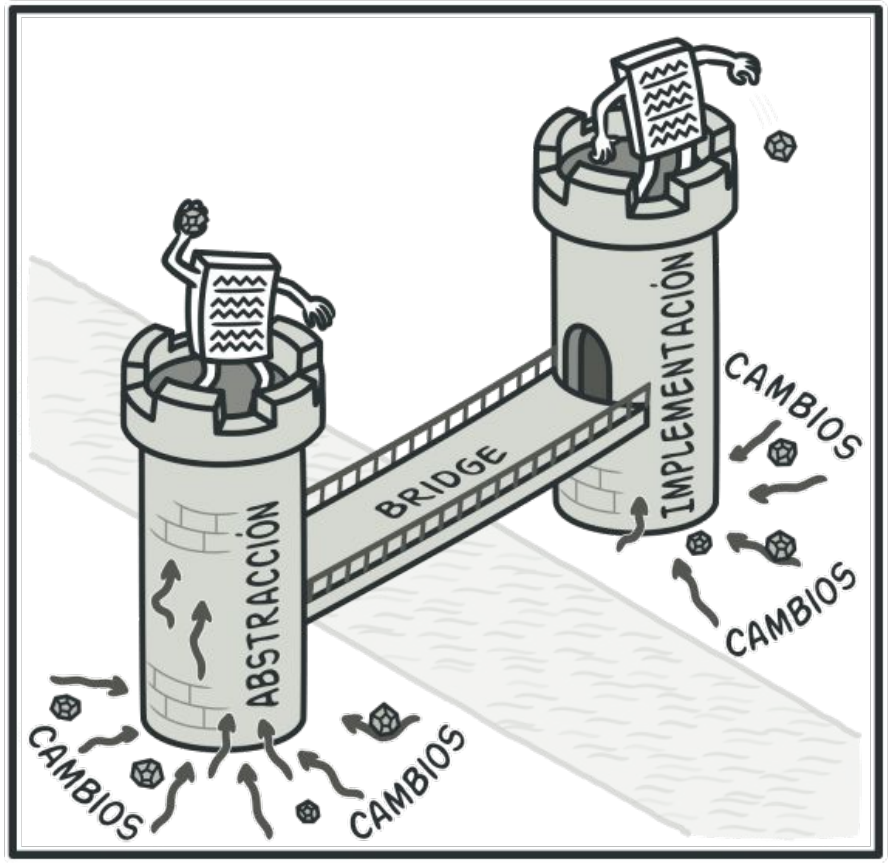
Bridge

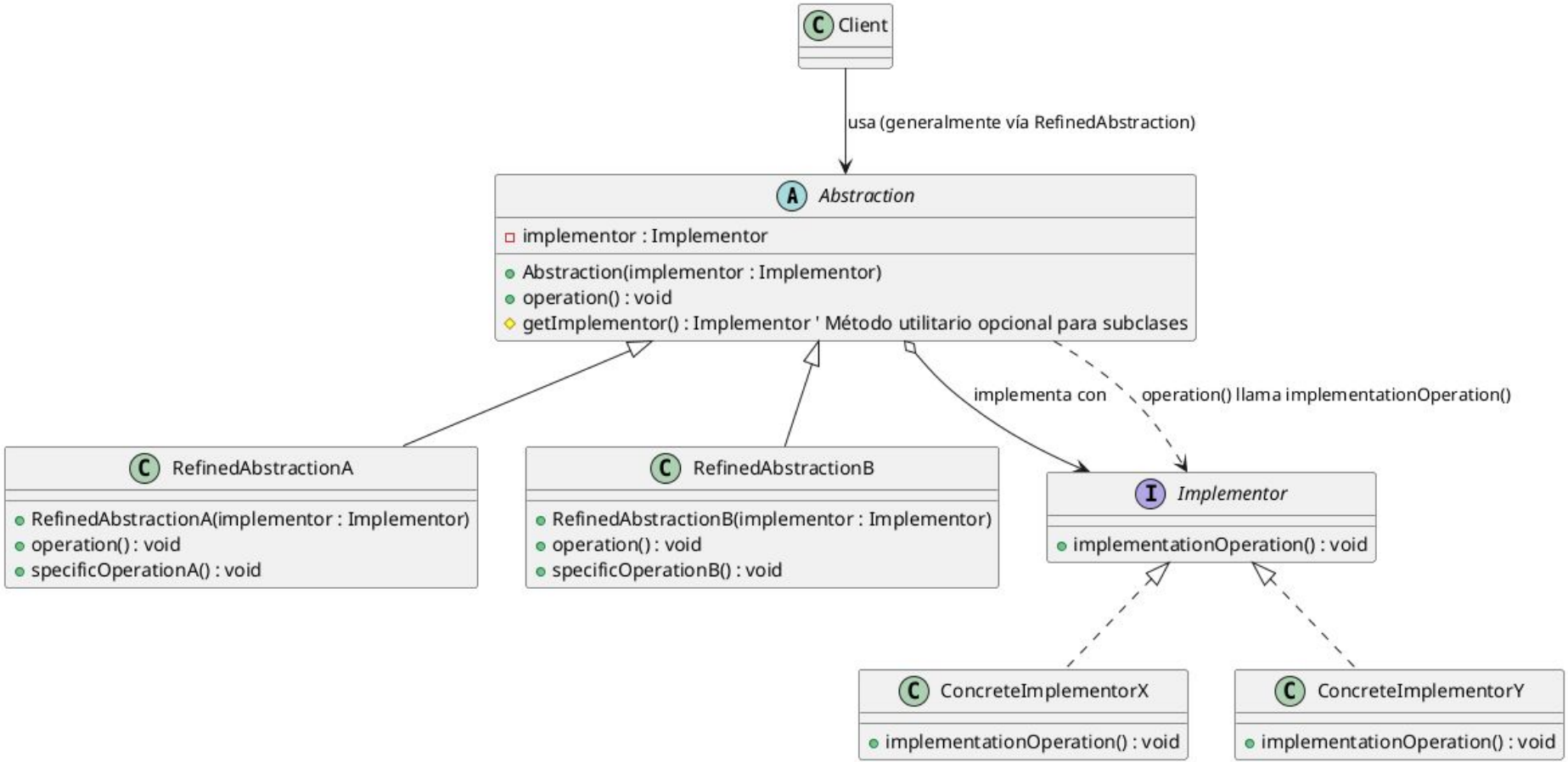


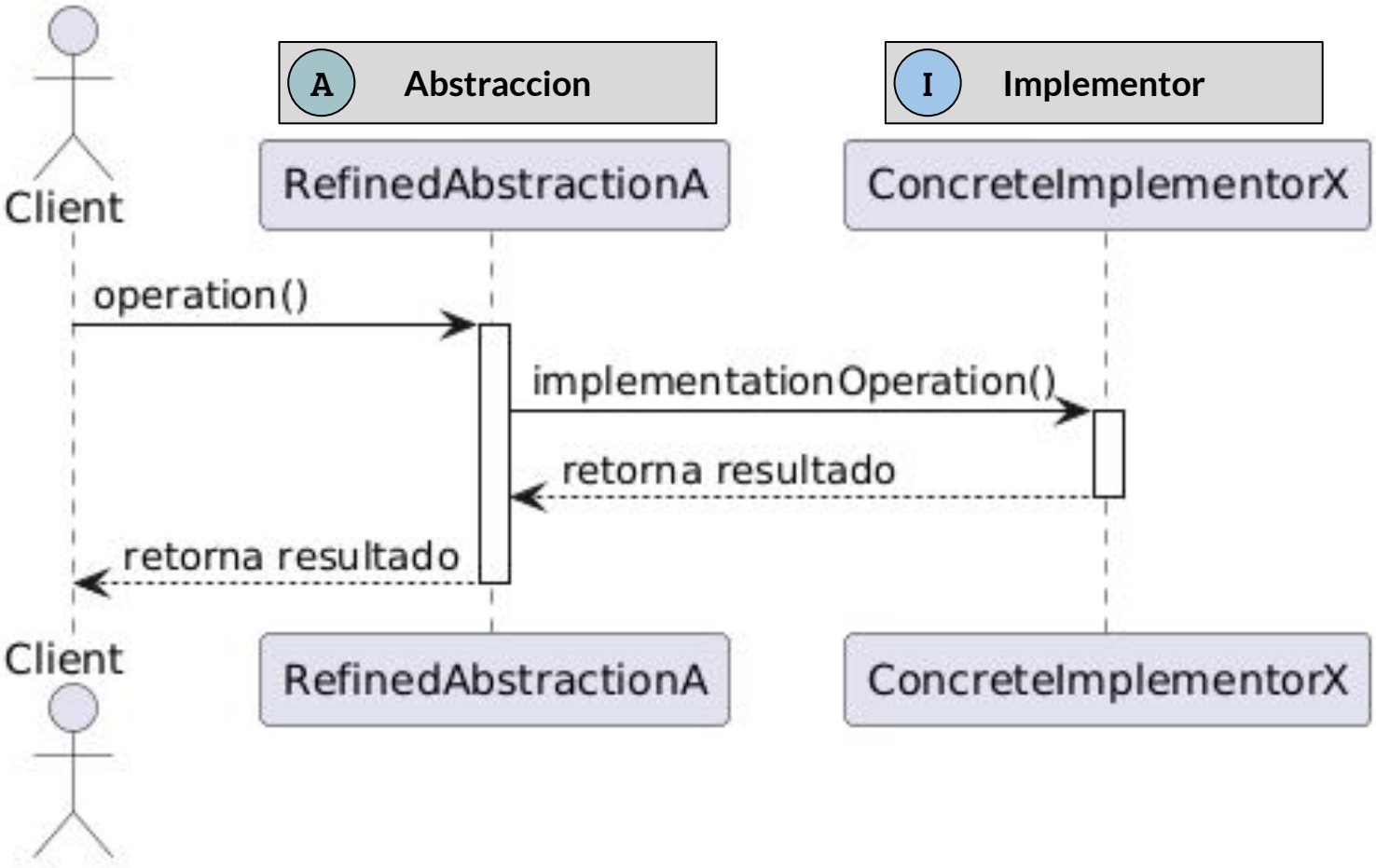
Bridge

desacopla una abstracción de su implementación, permitiendo que ambas varíen independientemente.


CAMBIOS
CAMBIOS
CAMBIOS
CAMBIOS
CAMBIOS
CAMBIOS

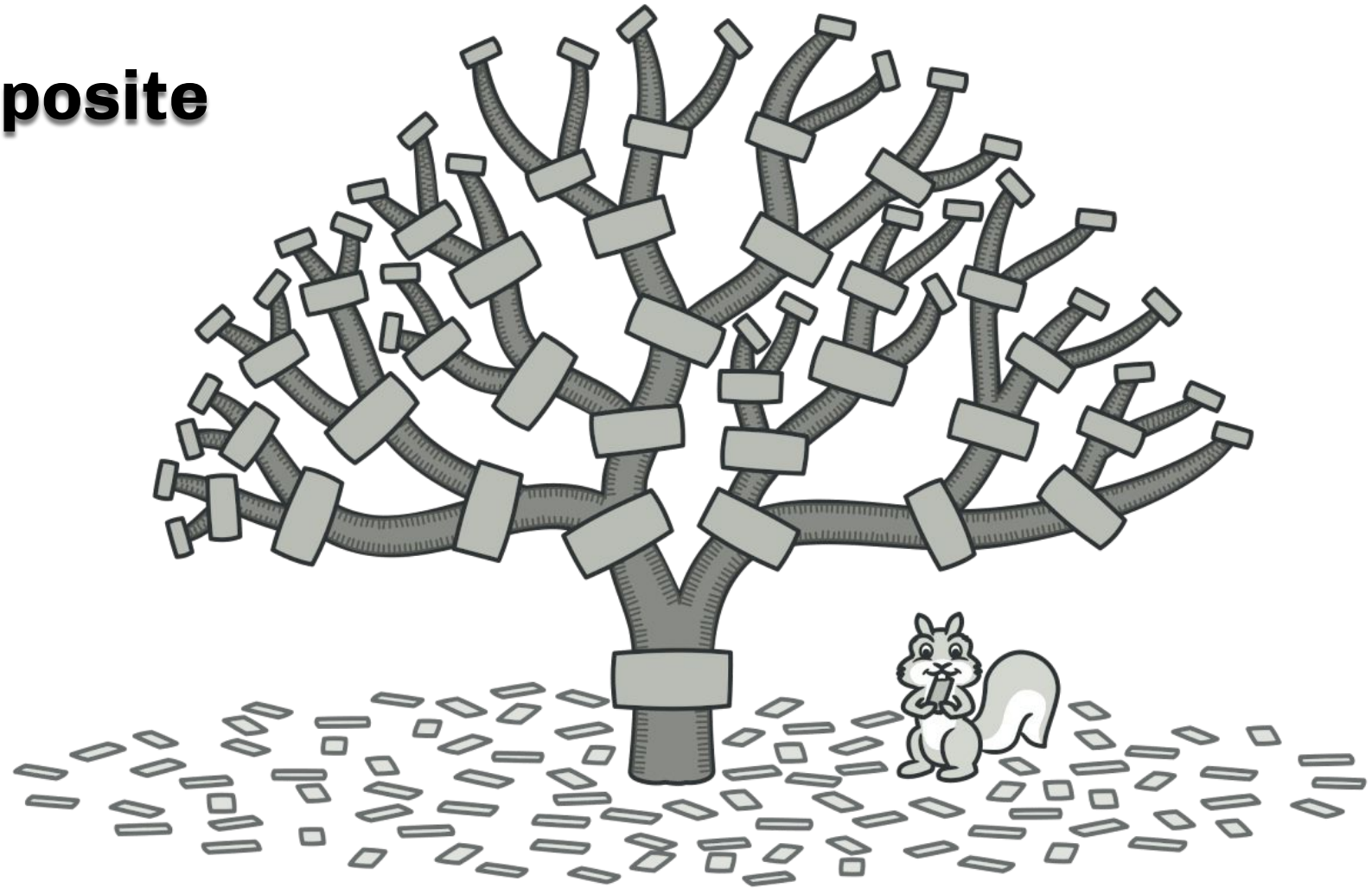

ABSTRACCIÓN
BRIDGE
IMPLEMENTACIÓN
CAMBIOS
CAMBIOS
CAMBIOS
CAMBIOS
CAMBIOS


Client
usa (generalmente vía RefinedAbstraction)
Abstraction
- implementor : Implementor
+ Abstraction(implementor : Implementor)
+ operation() : void
# getImplementor() : Implementor ' Método utilitario opcional para subclases
implementa con
operation() llama implementationOperation()
RefinedAbstractionA
+ RefinedAbstractionA(implementor : Implementor)
+ operation() : void
+ specificOperationA() : void
RefinedAbstractionB
+ RefinedAbstractionB(implementor : Implementor)
+ operation() : void
+ specificOperationB() : void
Implementor
+ implementationOperation() : void
ConcreteImplementorX
+ implementationOperation() : void
ConcreteImplementorY
+ implementationOperation() : void


A
Abstraccion
I
Implementor
Client
RefinedAbstractionA
ConcreteImplementorX
operation()
implementationOperation()
retorna resultado
retorna resultado
Client
RefinedAbstractionA
ConcreteImplementorX

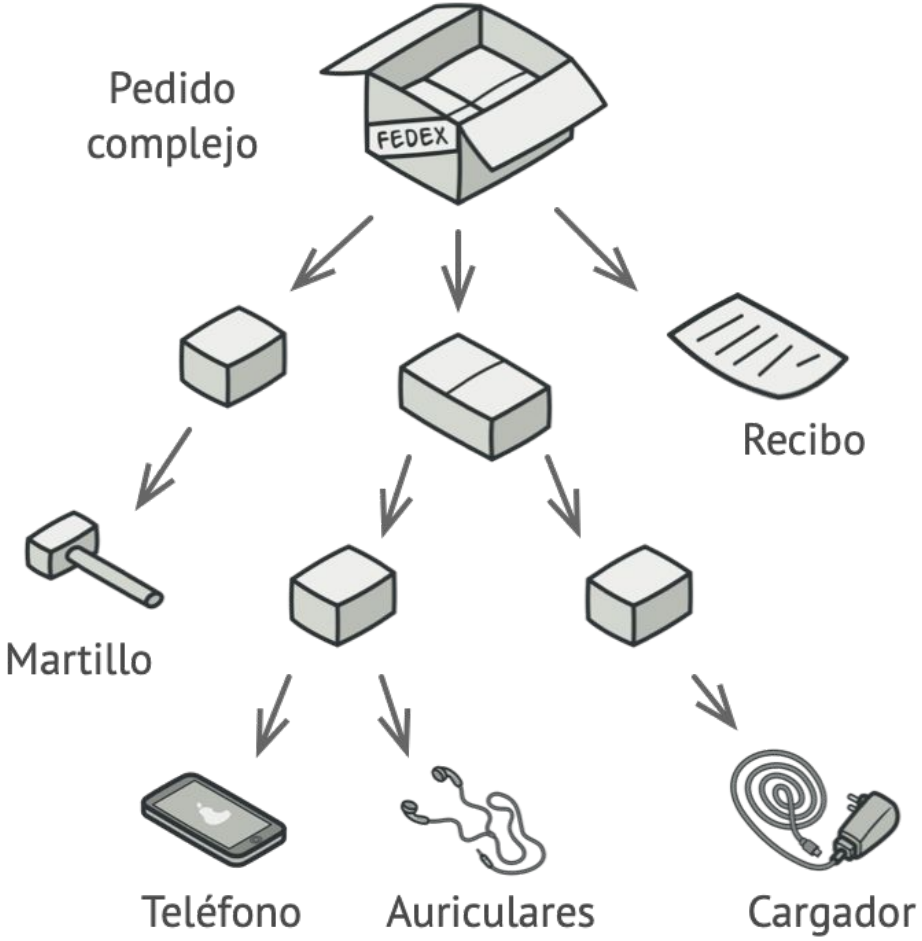

¿Preguntas?

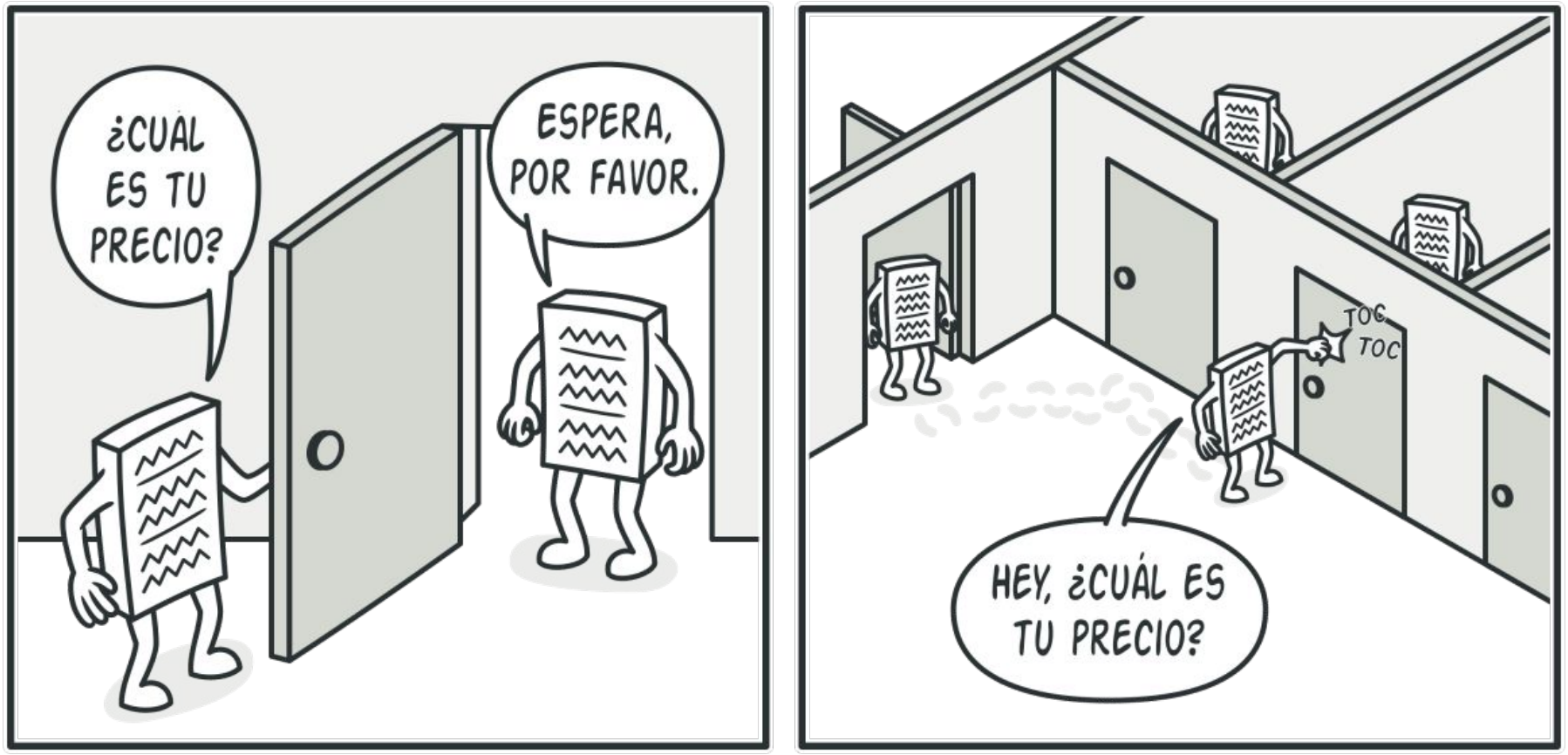
Composite

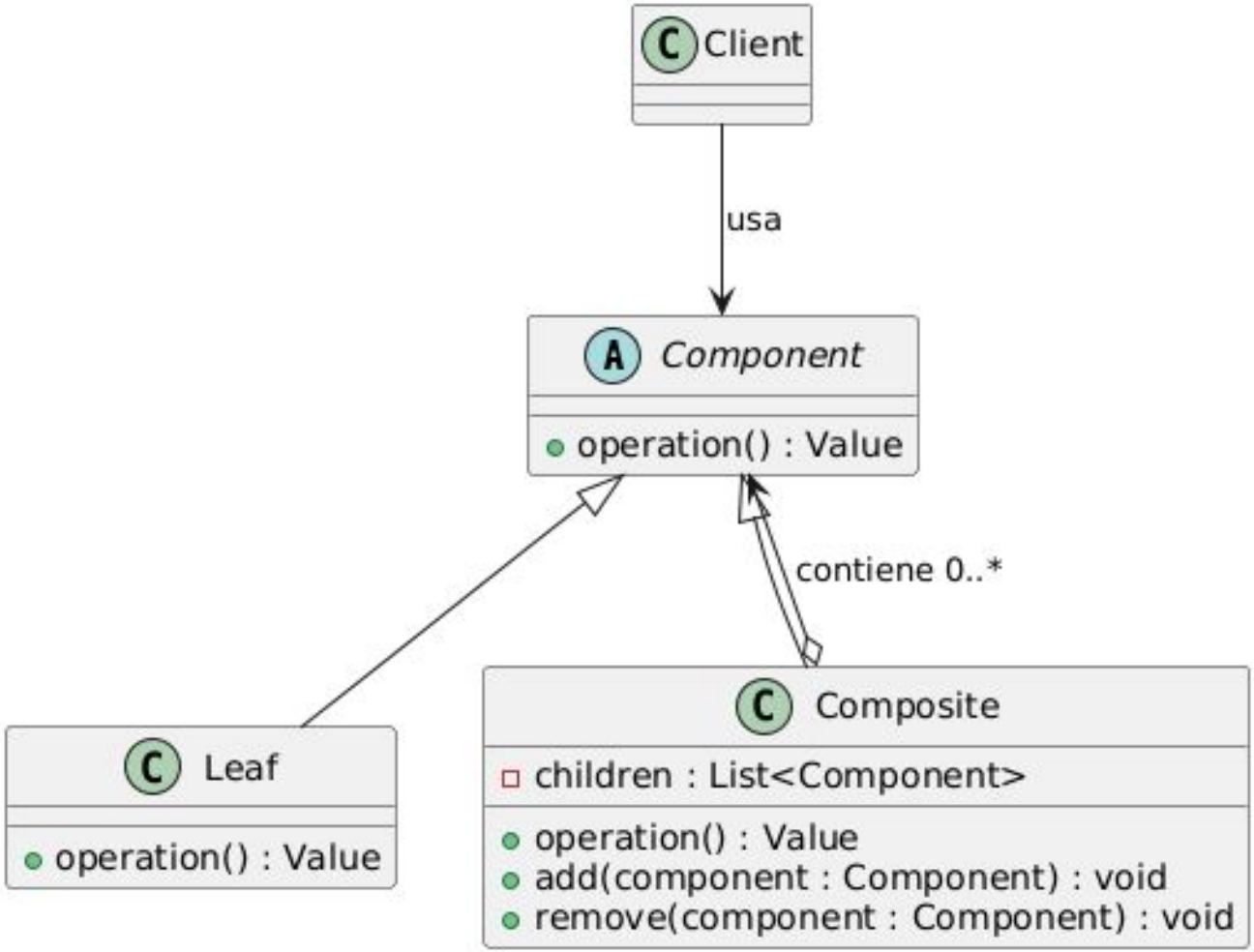


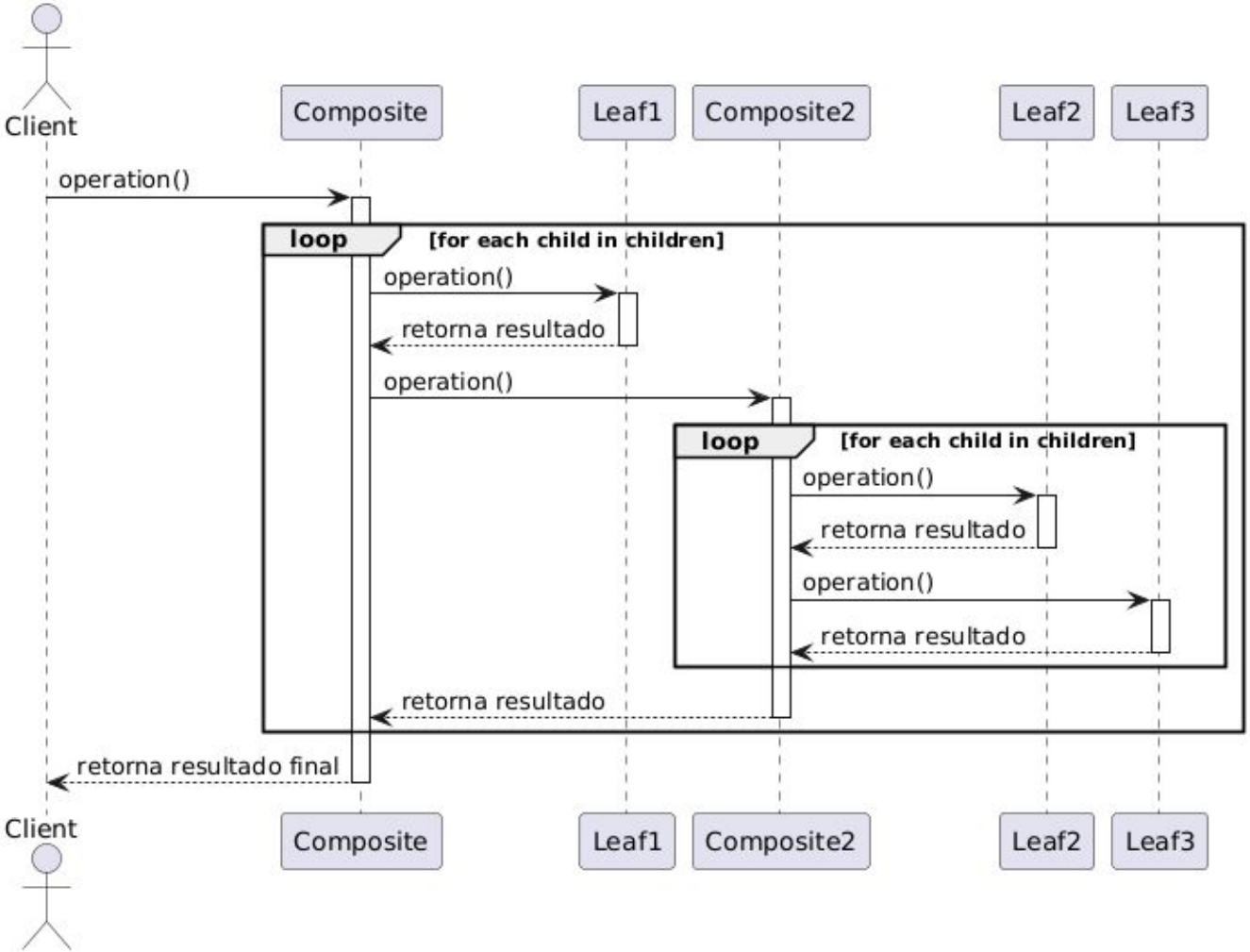
Composite

compone objetos en estructuras de árbol para representar jerarquías parte-todo, permitiendo a los clientes tratar objetos individuales y composiciones de objetos de manera uniforme

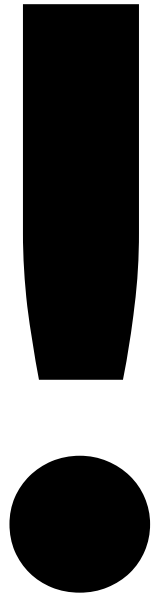

Pedido complejo
FEDEX
Recibo
Martillo
Teléfono
Auriculares
Cargador


¿CUÁL ES TU PRECIO?
ESPERA, POR FAVOR.
TOC TOC
HEY, ¿CUÁL ES TU PRECIO?


Client
usa
Component
operation() : Value
contiene 0..*
Leaf
operation() : Value
Composite
children : List<Component>
operation() : Value
add(component : Component) : void
remove(component : Component) : void


Client
Composite
Leaf1
Composite2
Leaf2
Leaf3
operation()
loop
[for each child in children]
operation()
retorna resultado
operation()
loop
[for each child in children]
operation()
retorna resultado
operation()
retorna resultado
retorna resultado
retorna resultado final
Client
Composite
Leaf1
Composite2
Leaf2
Leaf3

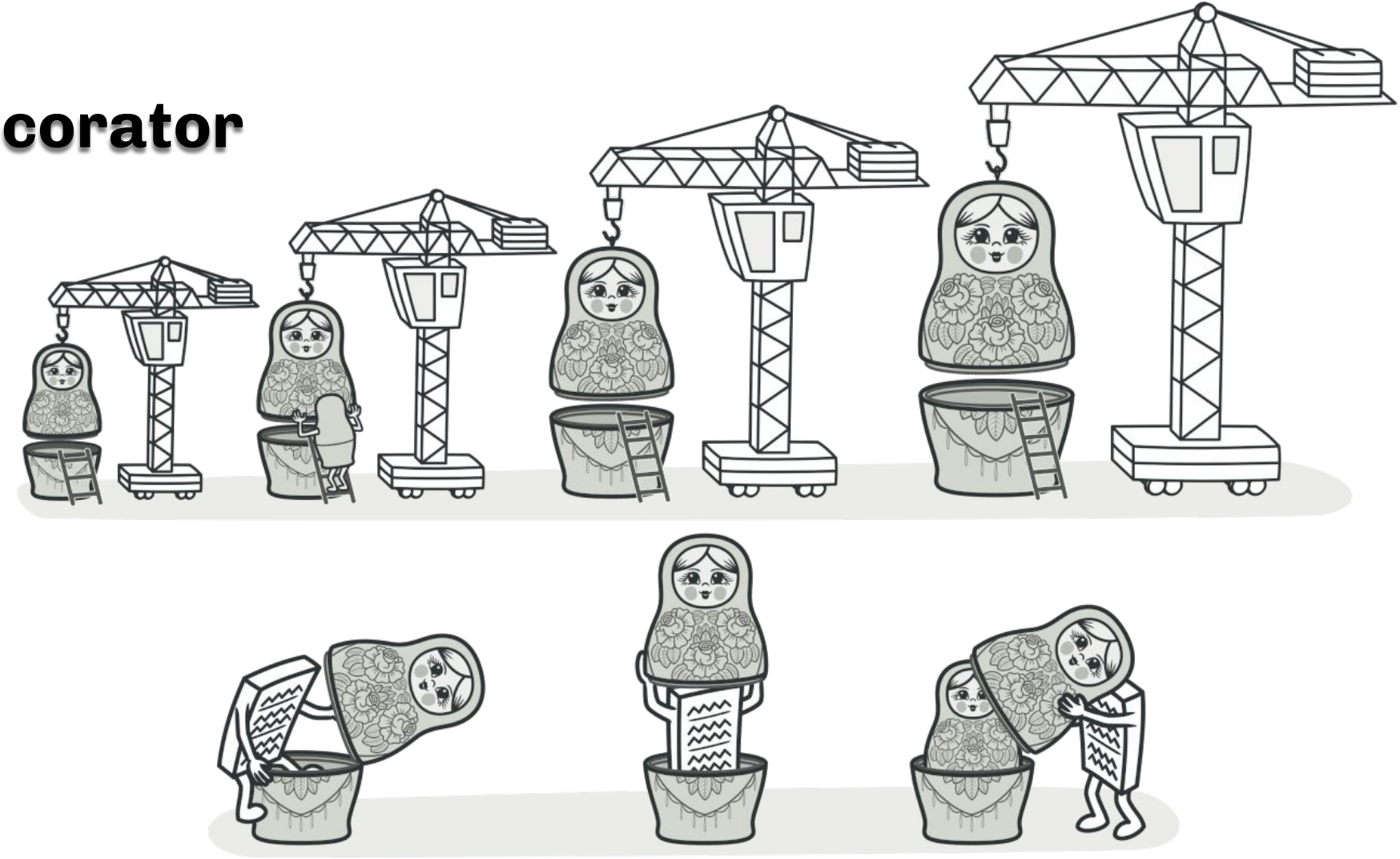
La calculadora



TP7

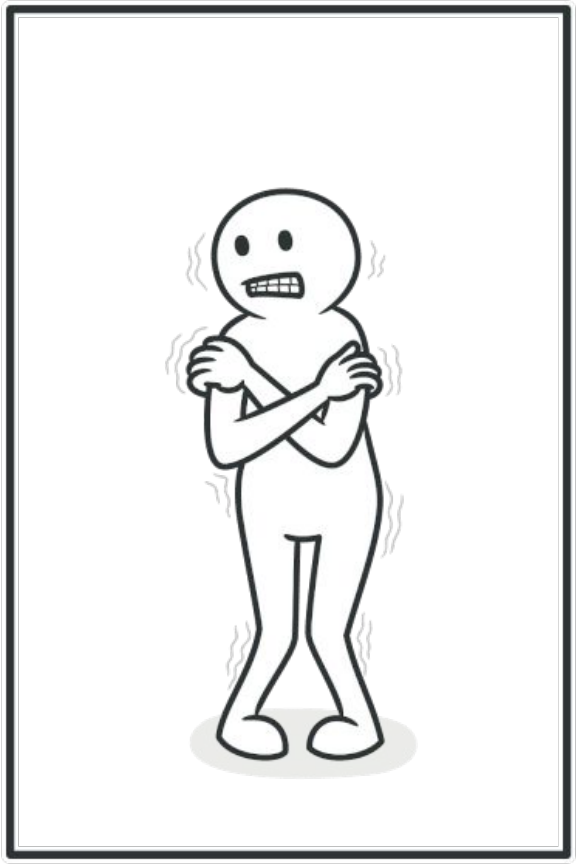

¿Preguntas?

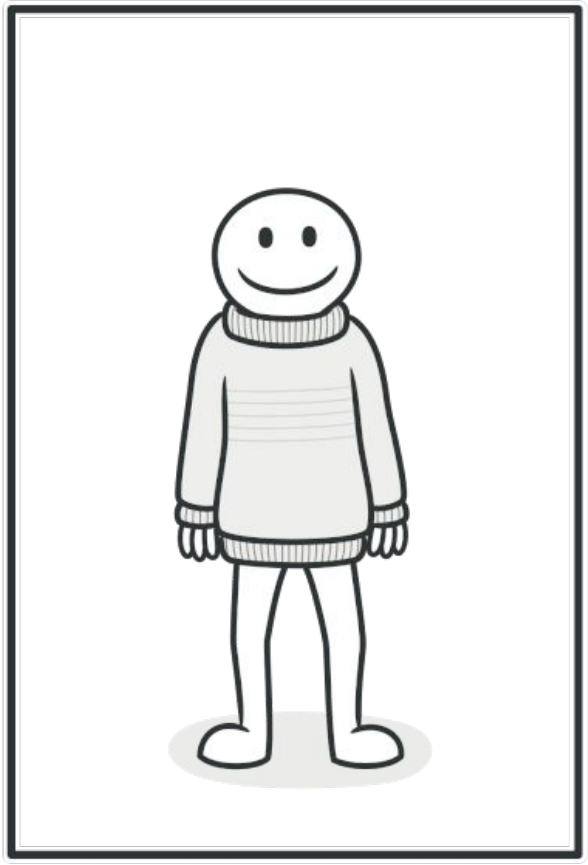
Decorator

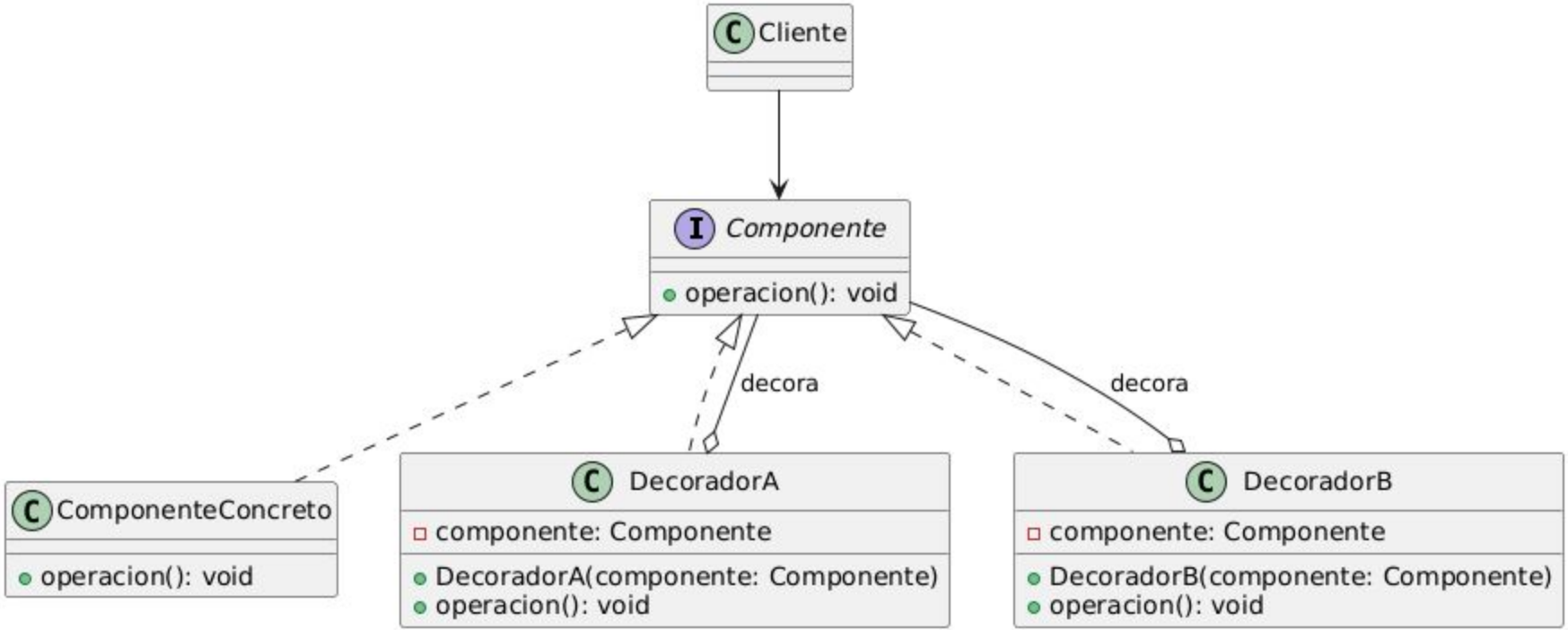


Decorator

adjunta responsabilidades adicionales a un objeto de forma dinámica y transparente envolviéndolo



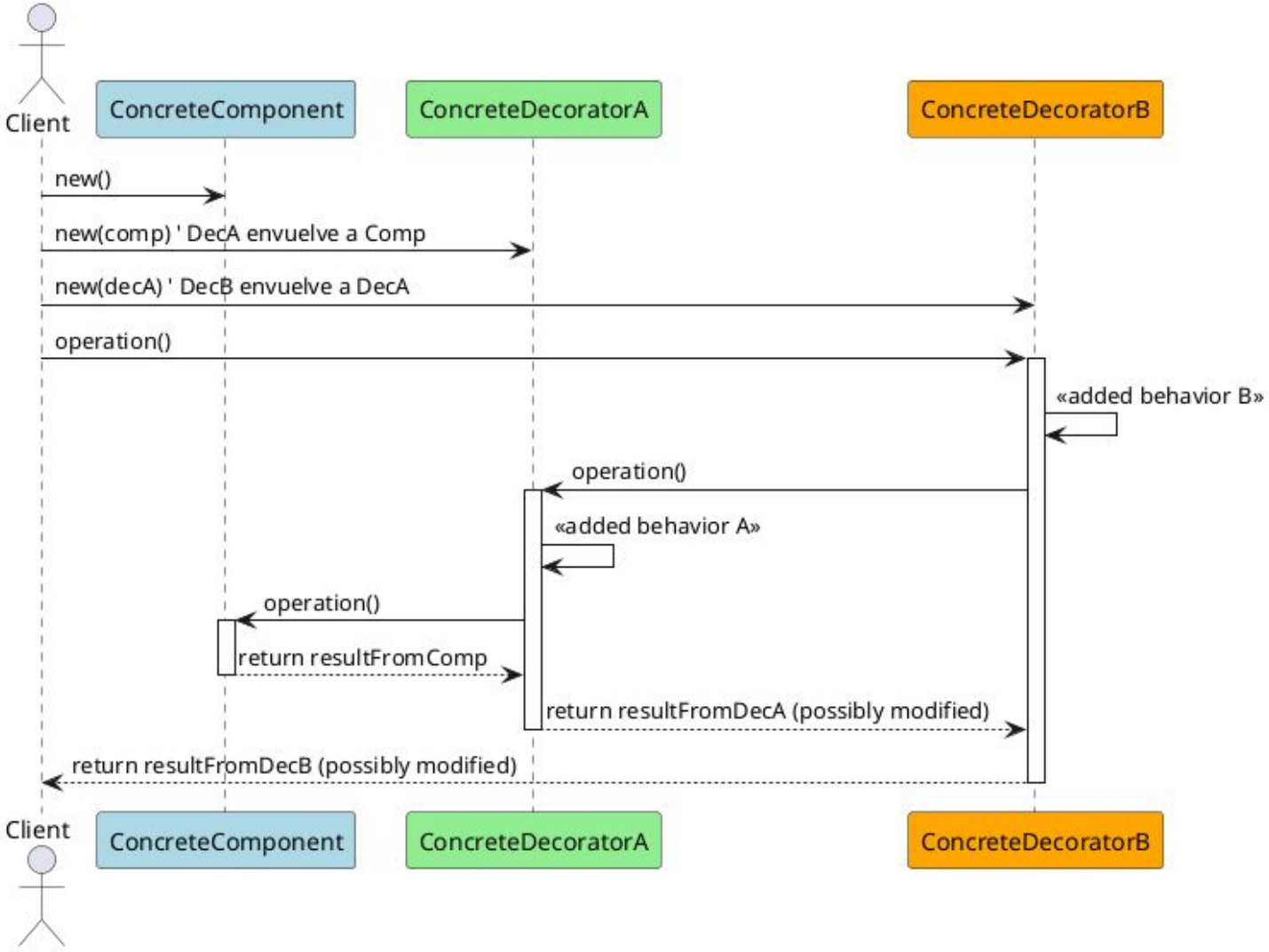



Cliente
Componente
operacion(): void
decora
decora
ComponenteConcreto
operacion(): void
DecoradorA
componente: Componente
DecoradorA(componente: Componente)
operacion(): void
DecoradorB
componente: Componente
DecoradorB(componente: Componente)
operacion(): void

En código termina siendo algo como...

```
Componente a = new DecoradorConcreto("2");
Componente c = new DecoradorA(new DecoradorConcreto("1"));
Componente d = new DecoradorB(c);

//y despues, operacion();
//  que se puede llamar en todos,
//  y con diferente comportamiento
```

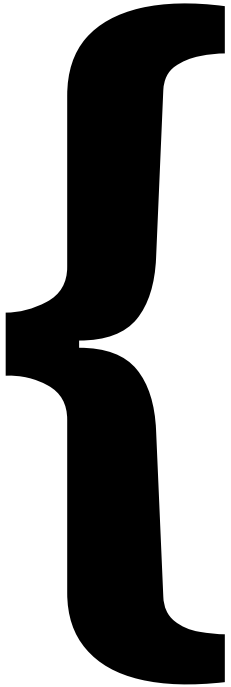

Client
ConcreteComponent
ConcreteDecoratorA
ConcreteDecoratorB
new()
new(comp) ' DecA envuelve a Comp
new(decA) ' DecB envuelve a DecA
operation()
«added behavior B»
operation()
«added behavior A»
operation()
return resultFromComp
return resultFromDecA (possibly modified)
return resultFromDecB (possibly modified)
Client
ConcreteComponent
ConcreteDecoratorA
ConcreteDecoratorB

{ Desde afuera, la interfaz no cambia }

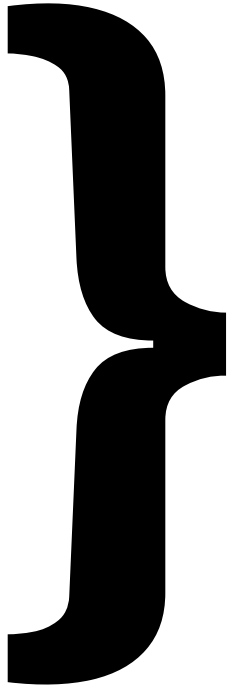
Esta Operacion , calcula solo cuando se lo pide.

```
public class CacheoOperacion extends Operacion {
private Operacion decorada;
long cacheo;
  public CacheoOperacion(Operacion aCachear){
    decorada = aCachear;
    refrescar();
  }
  public long calcular() {
    return cacheo;
  }
  public void refrescar() { cacheo = decorada.calcular(); }
}
```

Con una interfaz, bajamos el *acoplamiento conceptual* la

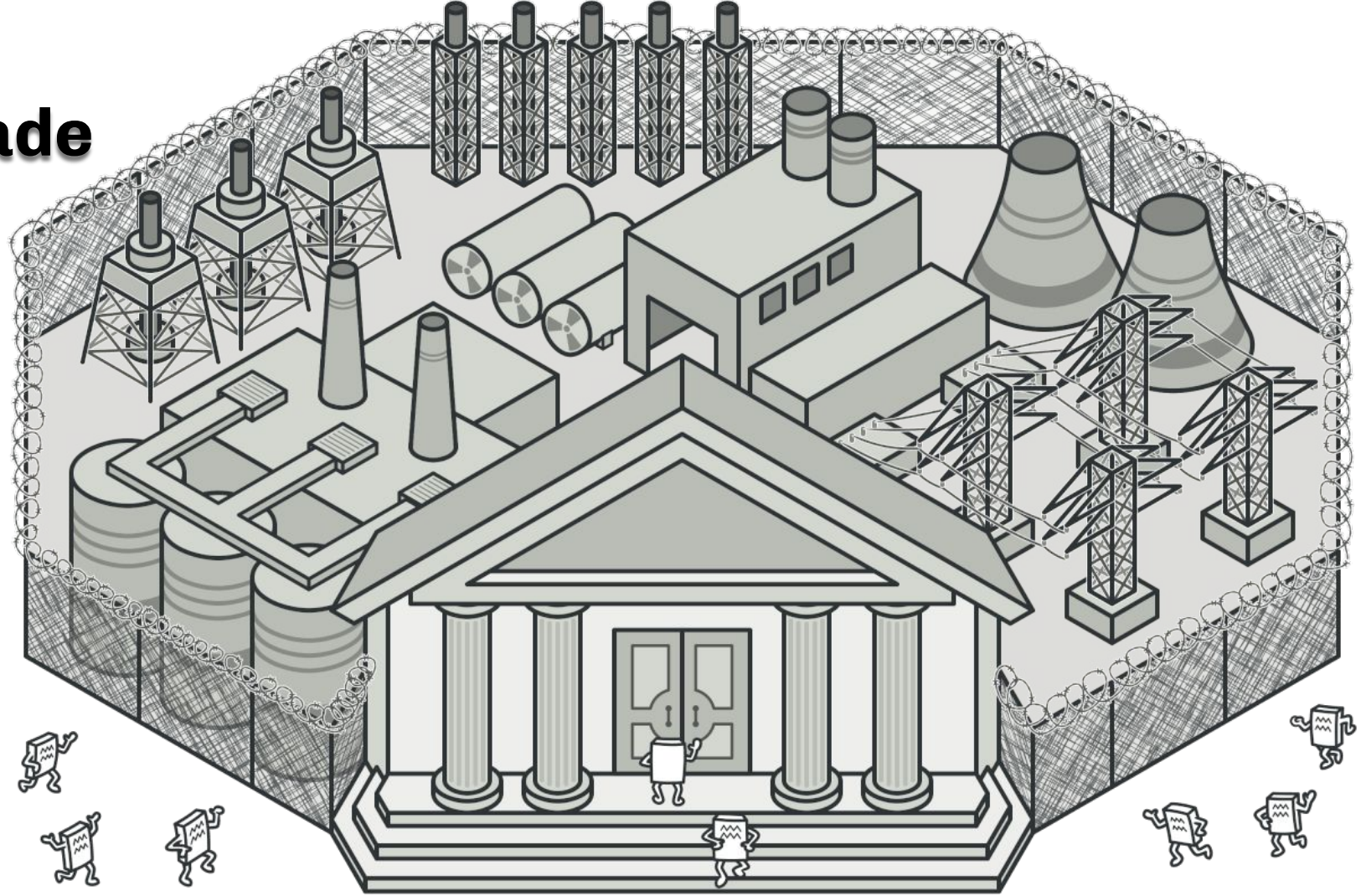


¡Funcionalidad opcional!



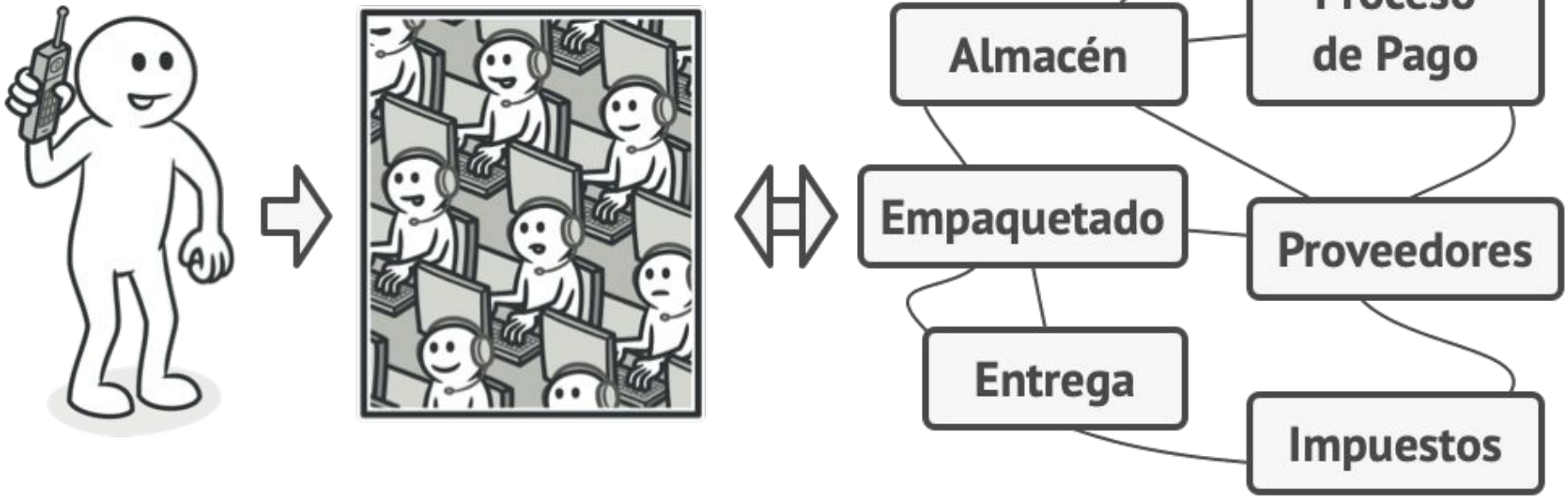

¿Preguntas?

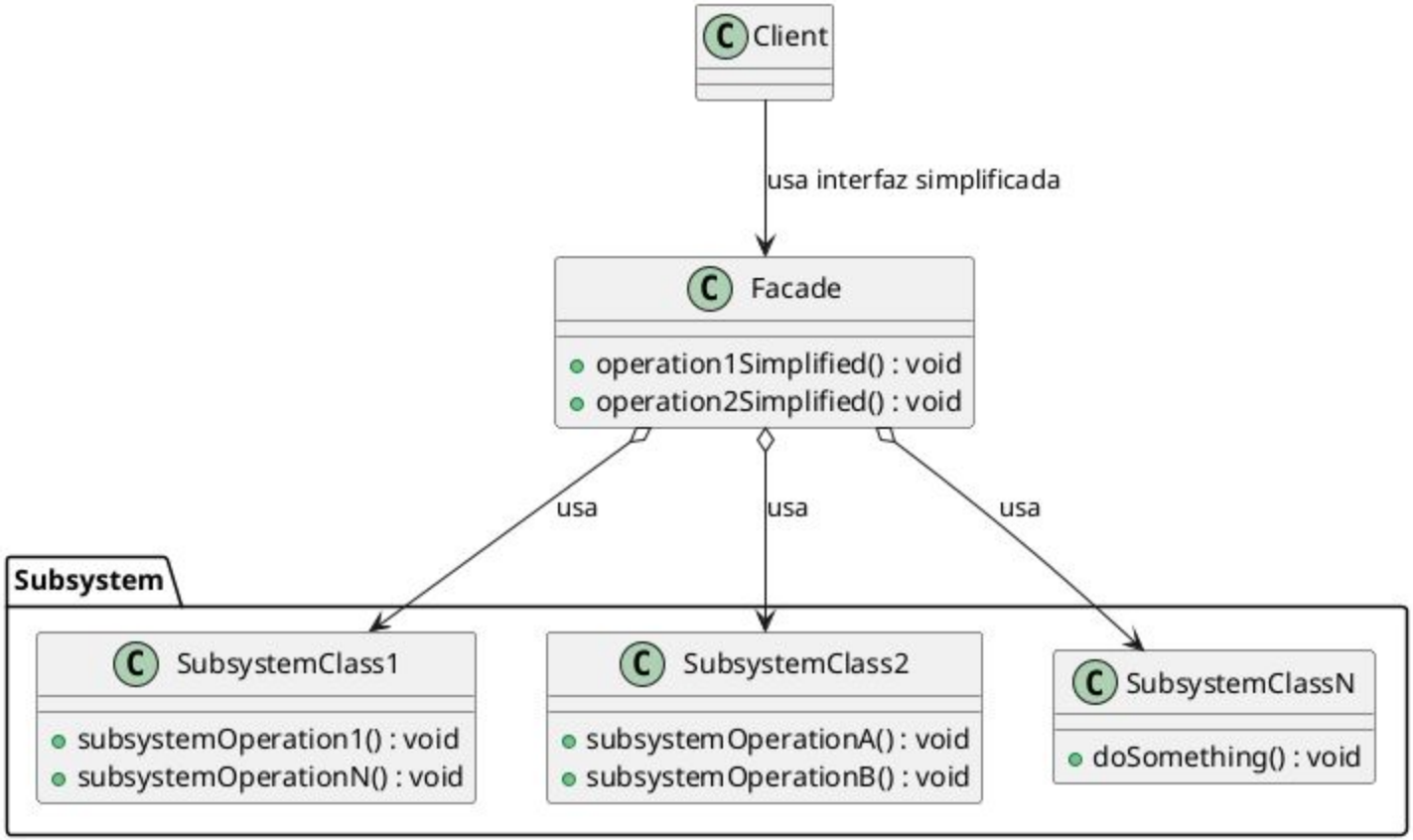
Facade

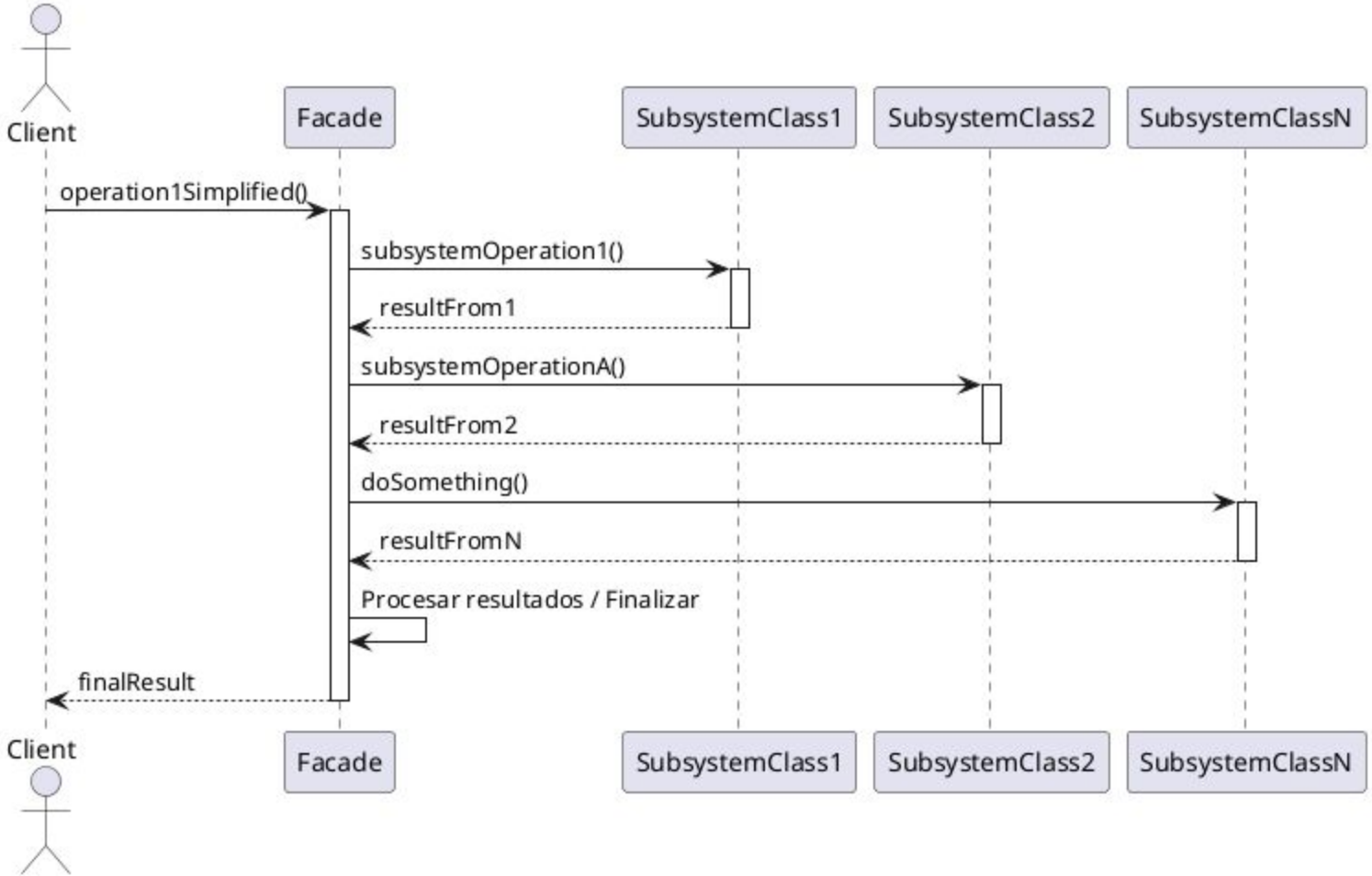


Facade

proporciona una interfaz unificada y simplificada a un conjunto de interfaces en un subsistema complejo.


Almacén
Proceso de Pago
Empaquetado
Proveedores
Entrega
Impuestos

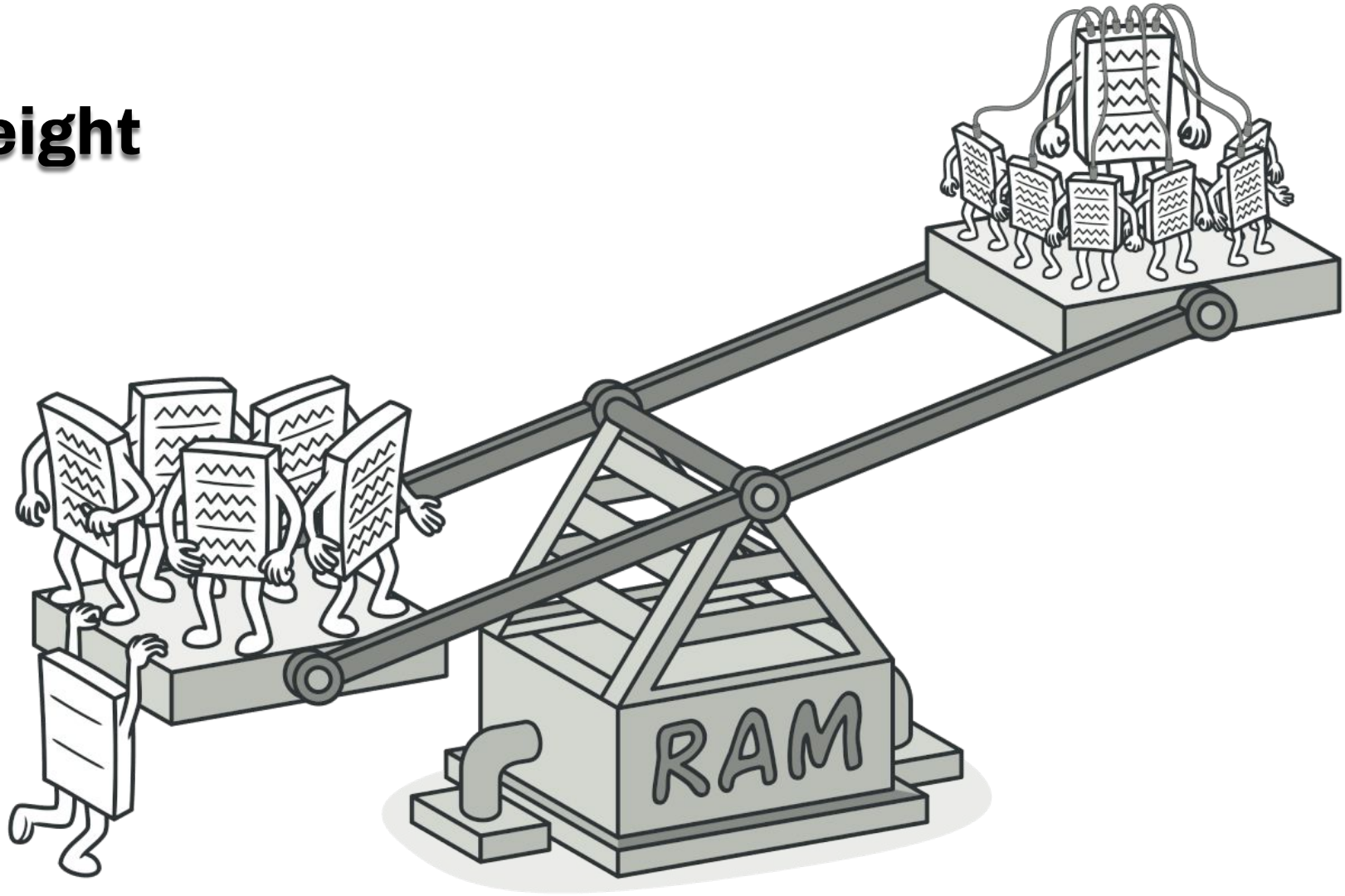

Client
usa interfaz simplificada
Facade
operation1Simplified() : void
operation2Simplified() : void
usa
usa
usa
Subsystem
SubsystemClass1
subsystemOperation1() : void
subsystemOperationN() : void
SubsystemClass2
subsystemOperationA() : void
subsystemOperationB() : void
SubsystemClassN
doSomething() : void


Client
Facade
SubsystemClass1
SubsystemClass2
SubsystemClassN
operation1Simplified()
subsystemOperation1()
resultFrom1
subsystemOperationA()
resultFrom2
doSomething()
resultFromN
Procesar resultados / Finalizar
finalResult
Client
Facade
SubsystemClass1
SubsystemClass2
SubsystemClassN

{ Similar a *Adapter* pero con otro objetivo general }

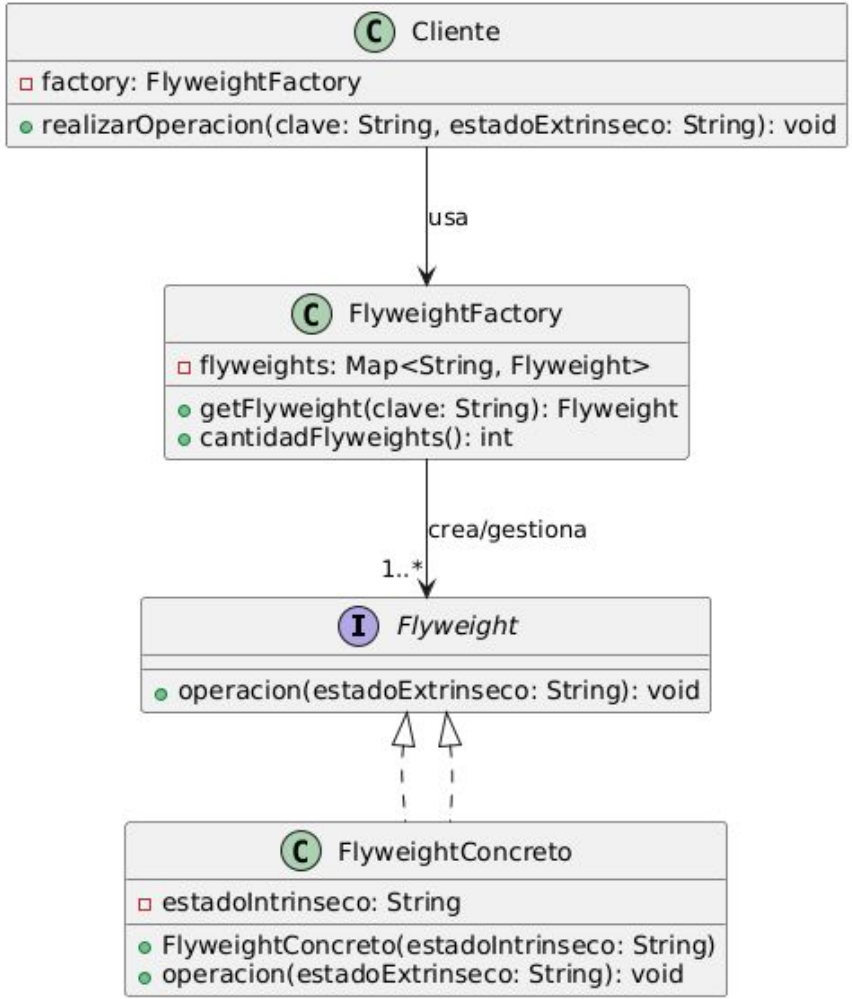

¿Preguntas?

Flyweight



Flyweight

gestiona datos compartidos para soportar grandes cantidades de objetos finos de manera eficiente.

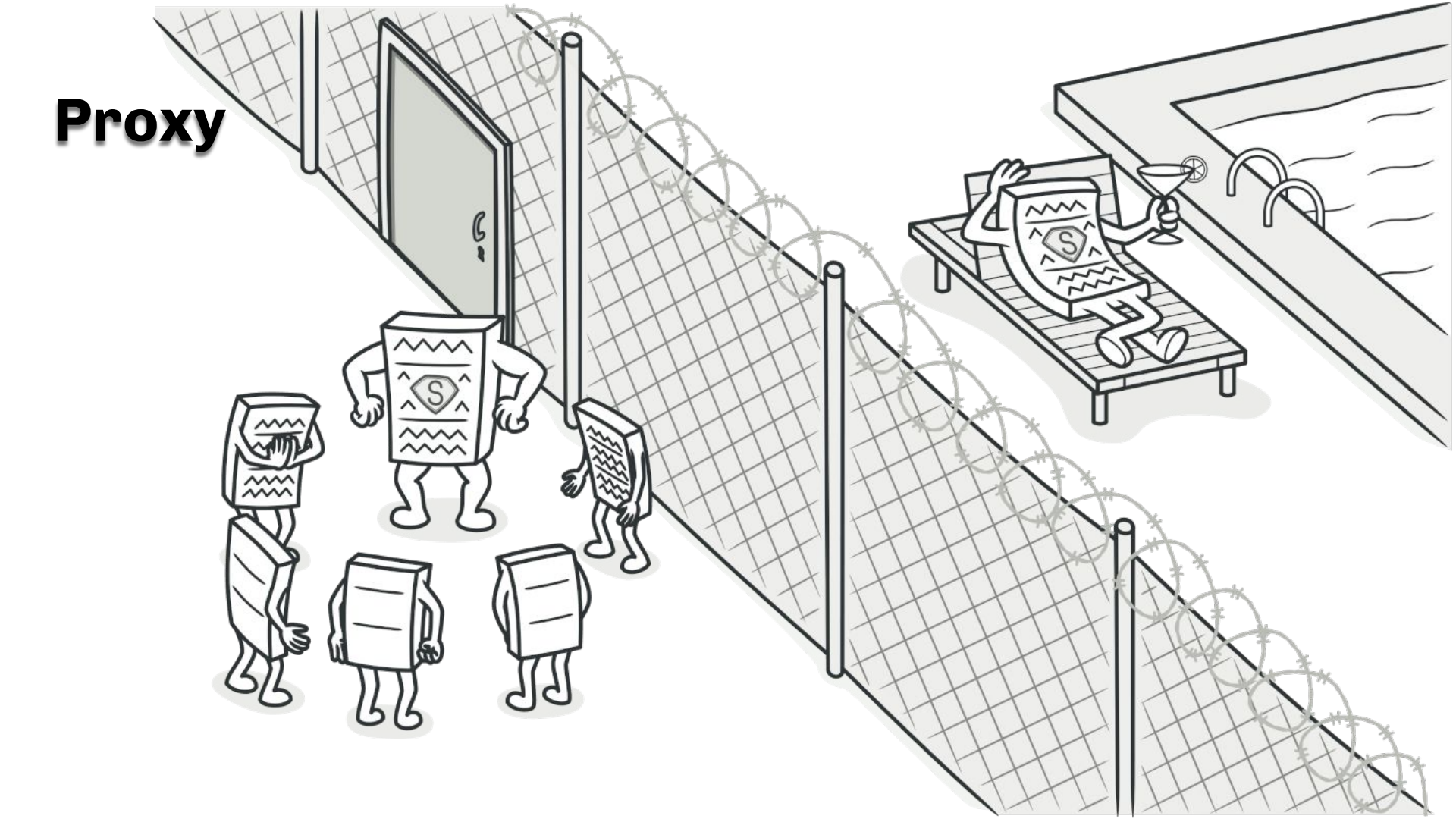

Cliente
- factory: FlyweightFactory
+ realizarOperacion(clave: String, estadoExtrinseco: String): void
usa
FlyweightFactory
- flyweights: Map<String, Flyweight>
+ getFlyweight(clave: String): Flyweight
+ cantidadFlyweights(): int
crea/gestiona
1..*
Flyweight
+ operacion(estadoExtrinseco: String): void
FlyweightConcreto
- estadoIntrinseco: String
+ FlyweightConcreto(estadoIntrinseco: String)
+ operacion(estadoExtrinseco: String): void

{ Permite reducir el consumo de recursos }

{ Lo compartido *debe* ser inmutable }

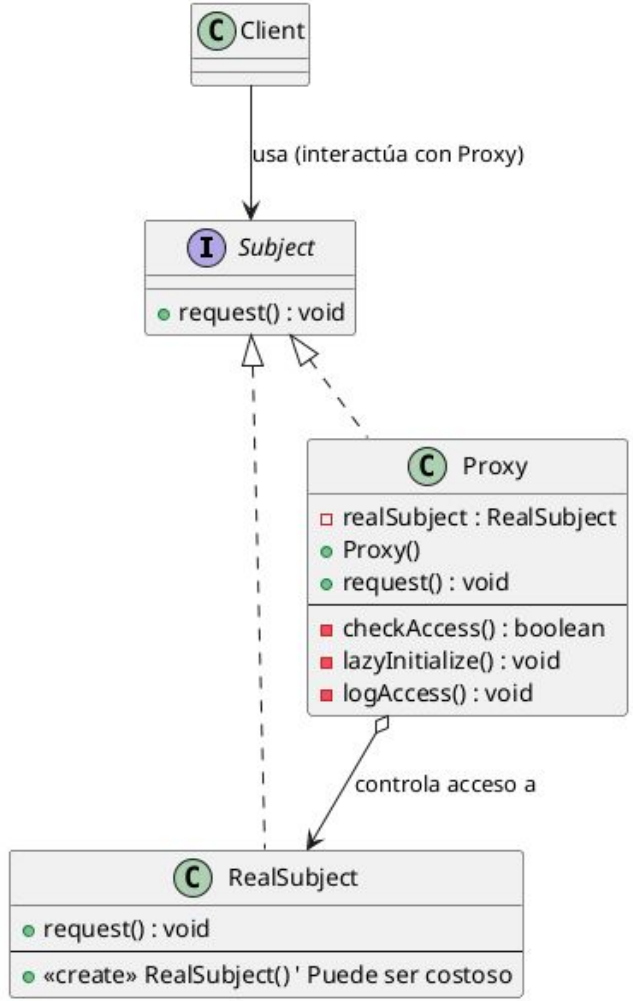

¿Preguntas?

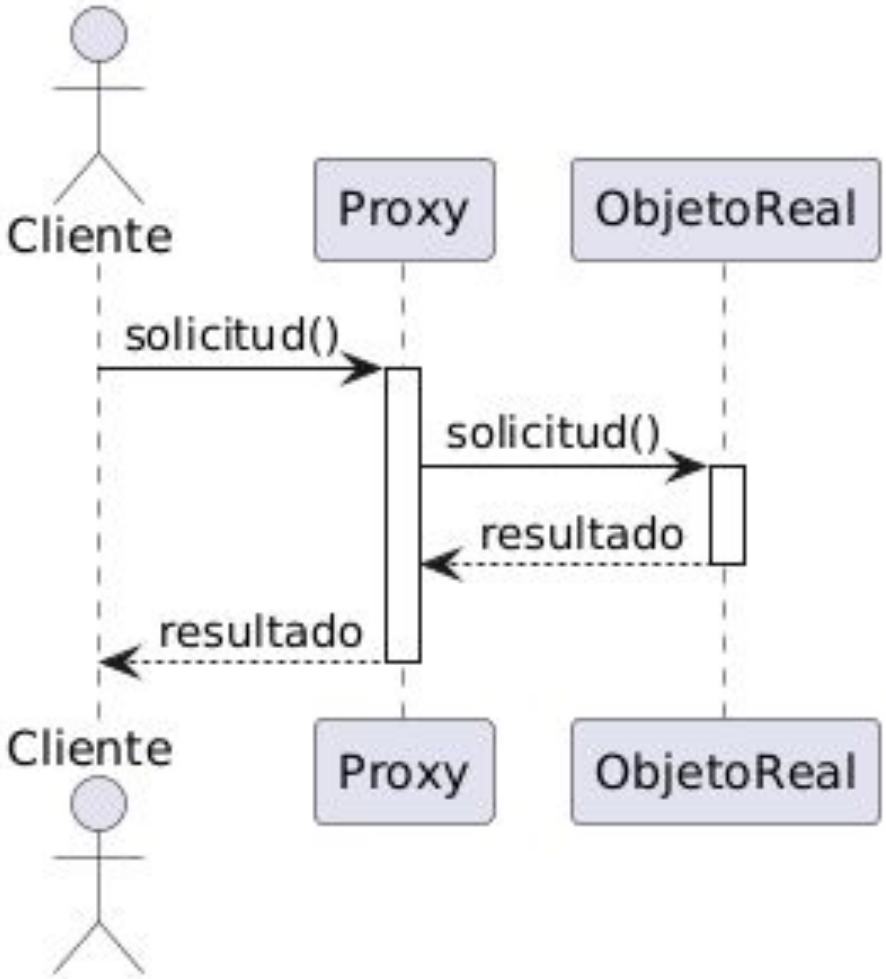
Proxy



Proxy

proporciona un intermediario para otro objeto a fin de controlar el acceso a este y sumar comportamientos.


Client
usa (interactúa con Proxy)
Subject
request() : void
Proxy
realSubject : RealSubject
Proxy()
request() : void
checkAccess() : boolean
lazyInitialize() : void
logAccess() : void
controla acceso a
RealSubject
request() : void
«create» RealSubject() ' Puede ser costoso


Cliente
Proxy
ObjetoReal
solicitud()
solicitud()
resultado
resultado
Cliente
Proxy
ObjetoReal

Naturalmente combinable con Decorator

Algunos usos de Proxy

Proxy Virtual

Controla la creación del Sujeto Real, postergándola hasta que realmente se necesita (carga perezosa). Útil para objetos costosos de crear.

Protección

Controla el acceso al Sujeto Real basándose en los permisos del cliente que realiza la solicitud.

Remoto

Gestiona los detalles de la comunicación de red para enviar y recibir solicitudes y resultados de una instancia en otro proceso.

Registro

Registra las llamadas a métodos y sus argumentos antes de delegar al Sujeto Real.

Caché

Almacena en caché los resultados de operaciones costosas y devuelve resultados almacenados para solicitudes repetidas sin delegar al Sujeto Real.

Proxy vs. Decorator

Proxy

Controlar el acceso a un objeto. Puede usarse para agregar lógica adicional antes o después de acceder al objeto real.

Decorator

Añadir responsabilidades o funcionalidades adicionales a un objeto de forma dinámica.

TP10

Arreglos + Patrones


¿Preguntas?

unrn.edu.ar

UNRN Universidad Nacional de Río Negro

| unrionegro