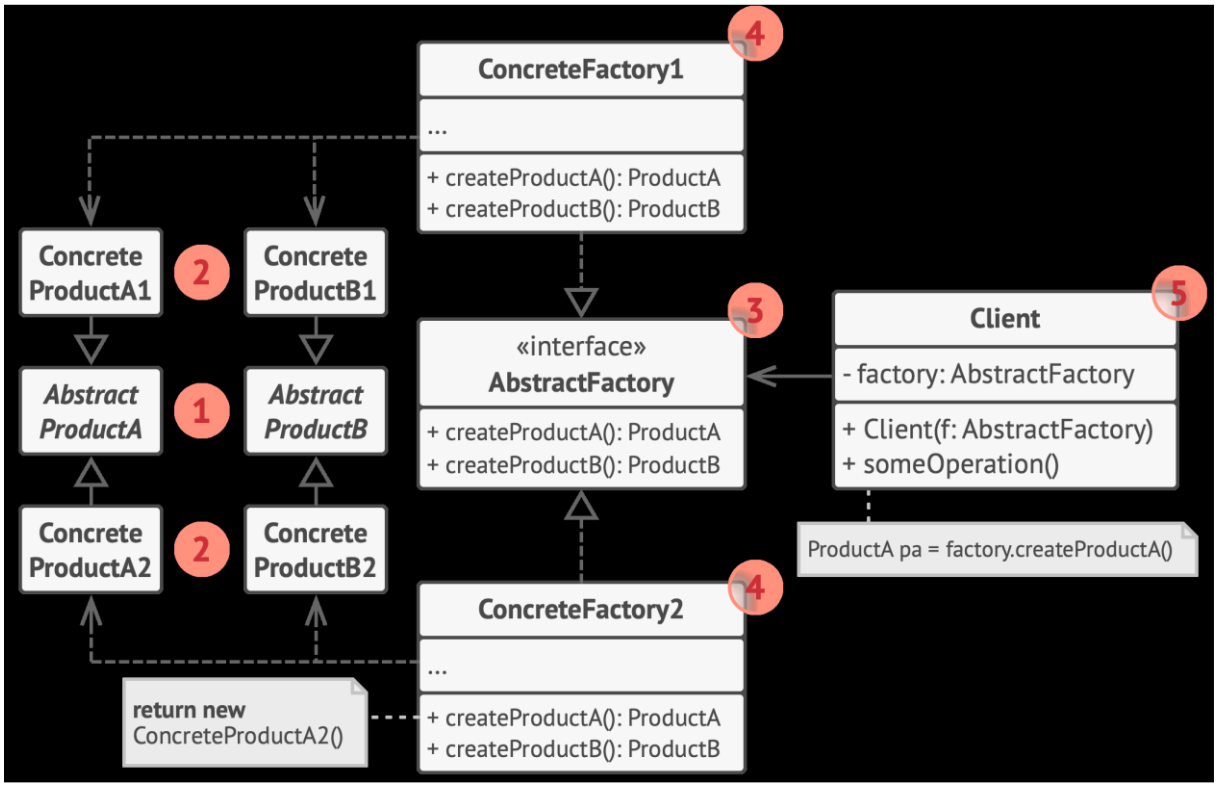
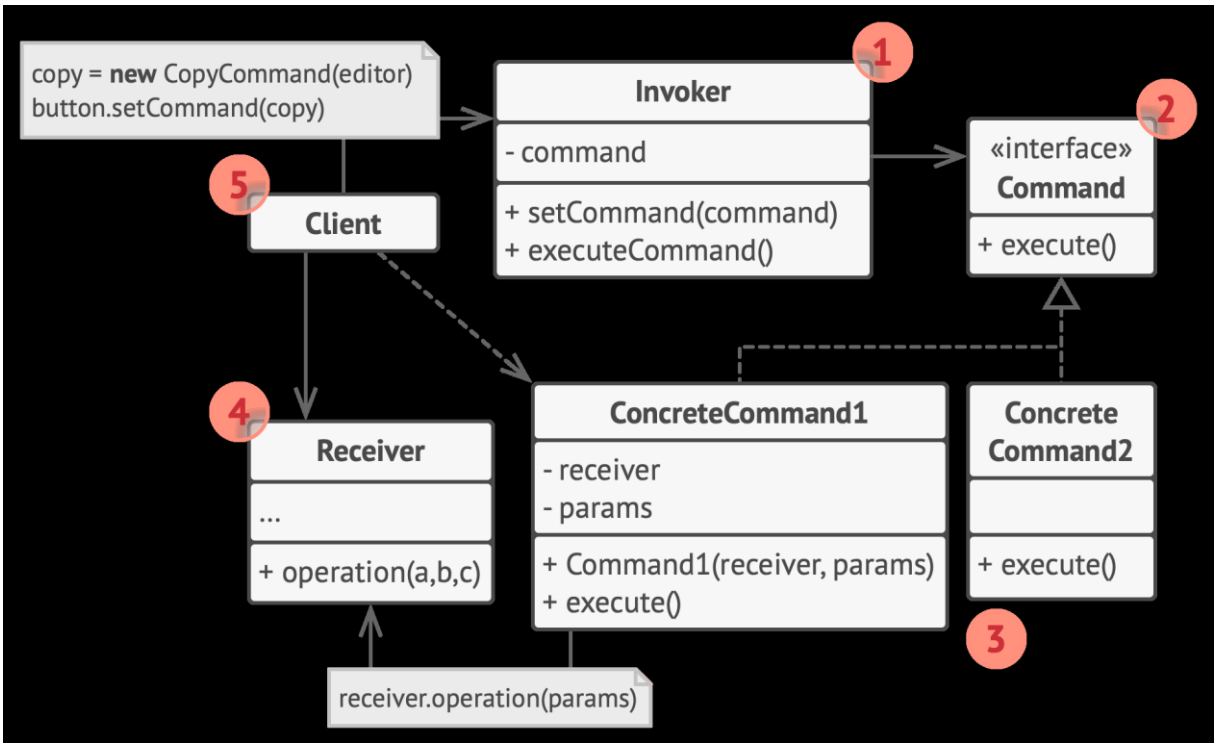
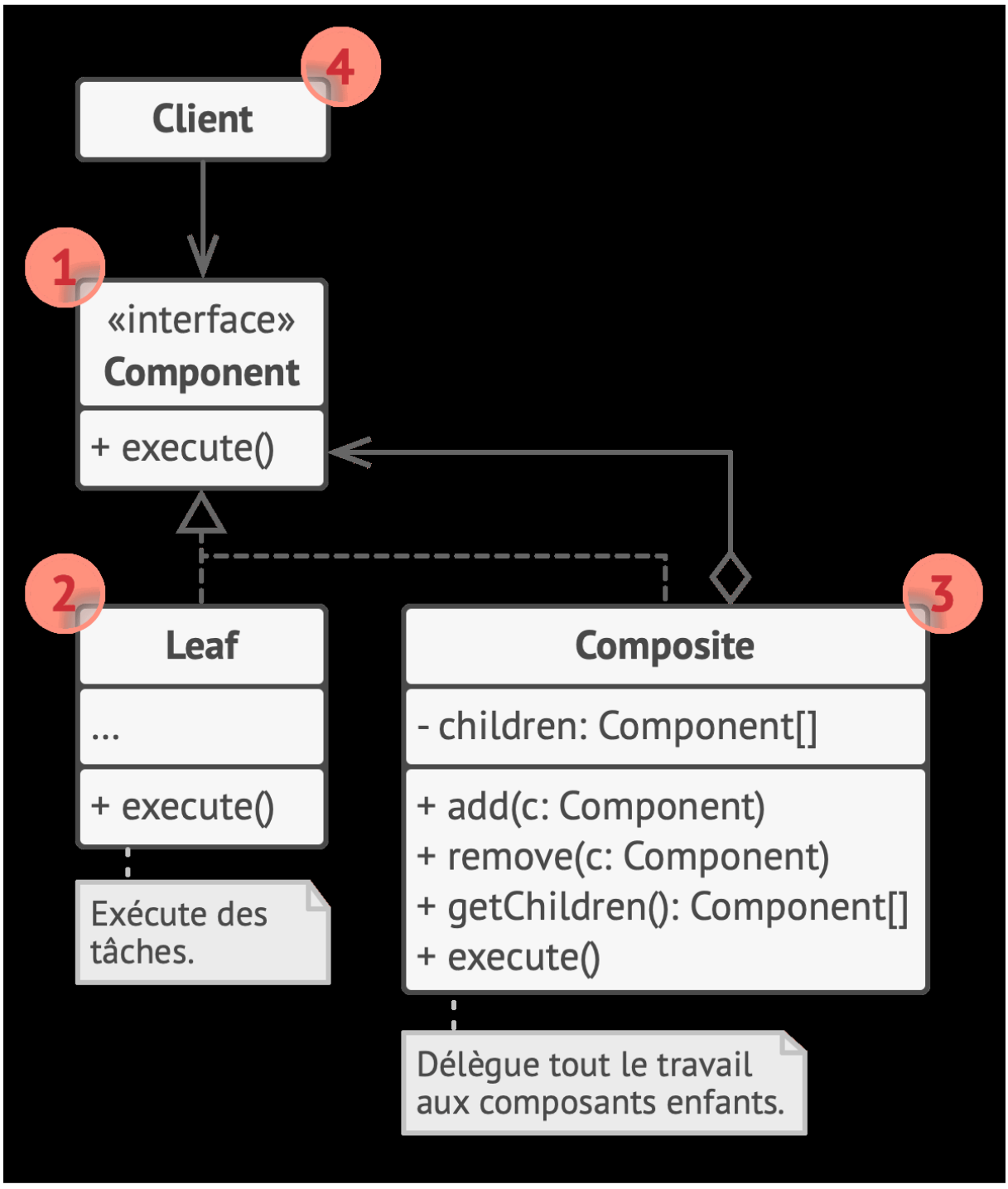
AbstractFactory



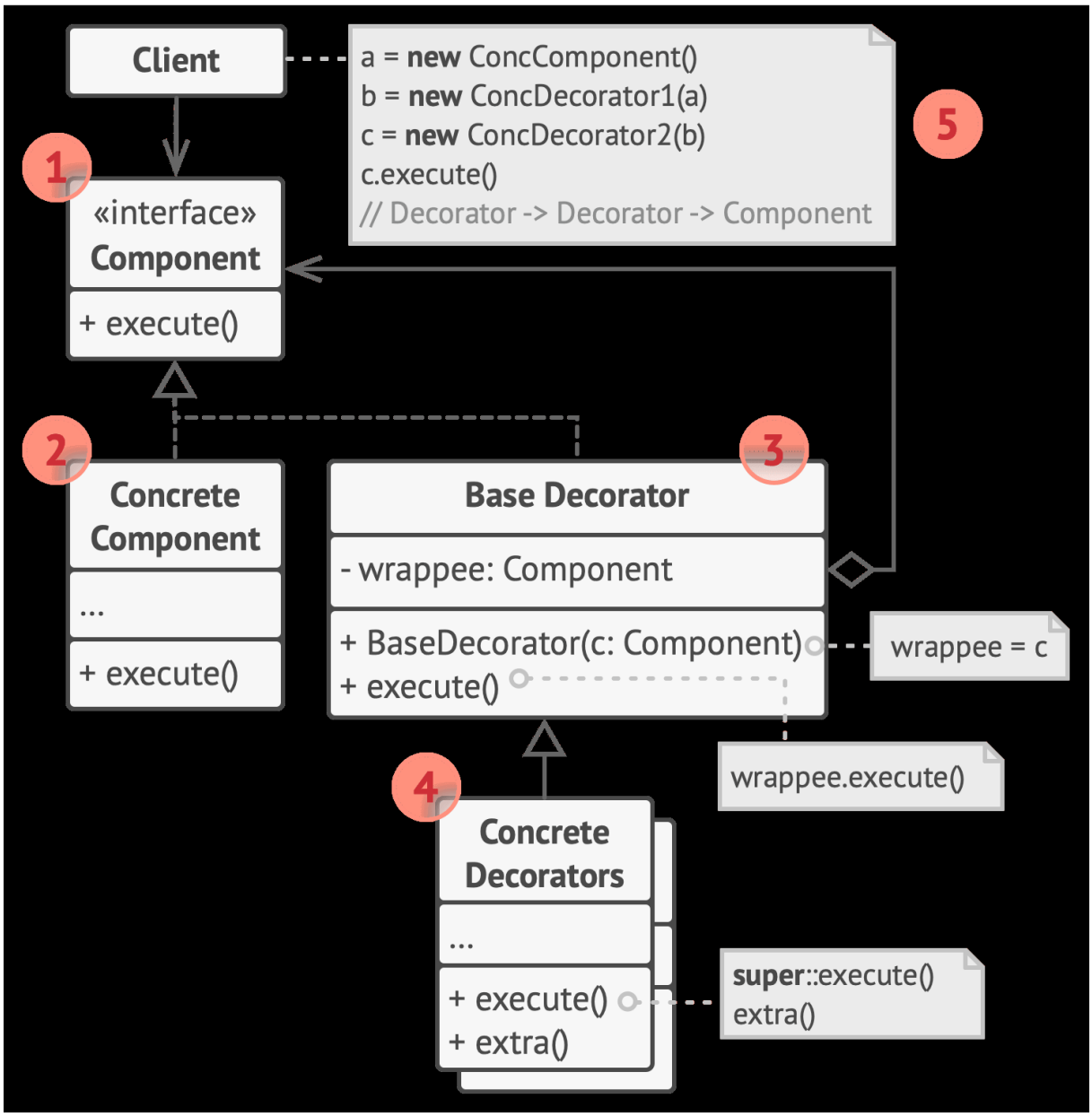
Command


copy = new CopyCommand(editor)
button.setCommand(copy)
Invoker
- command
+ setCommand(command)
+ executeCommand()
«interface»
Command
+ execute()
Client
Receiver
...
+ operation(a,b,c)
ConcreteCommand1
- receiver
- params
+ Command1(receiver, params)
+ execute()
Concrete
Command2
+ execute()
receiver.operation(params)
1
2
3
4
5

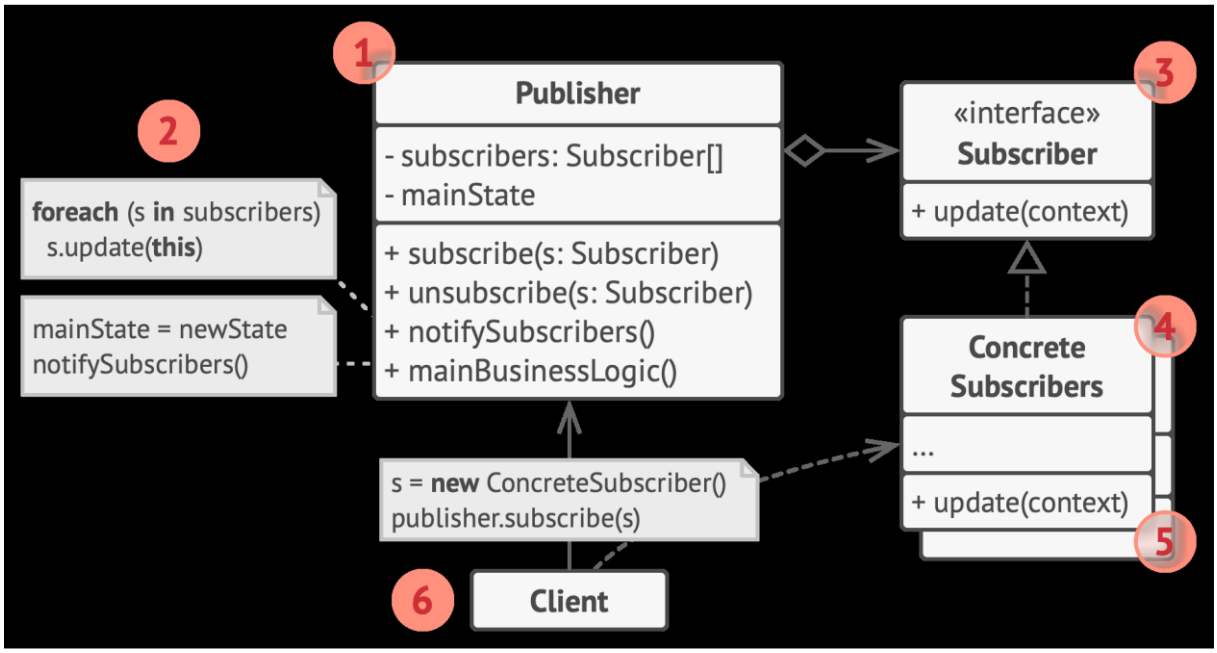
Composite


4
Client
1
«interface»
Component
+ execute()
2
Leaf
...
+ execute()
Exécute des tâches.
3
Composite
- children: Component[]
+ add(c: Component)
+ remove(c: Component)
+ getChildren(): Component[]
+ execute()
Délègue tout le travail aux composants enfants.

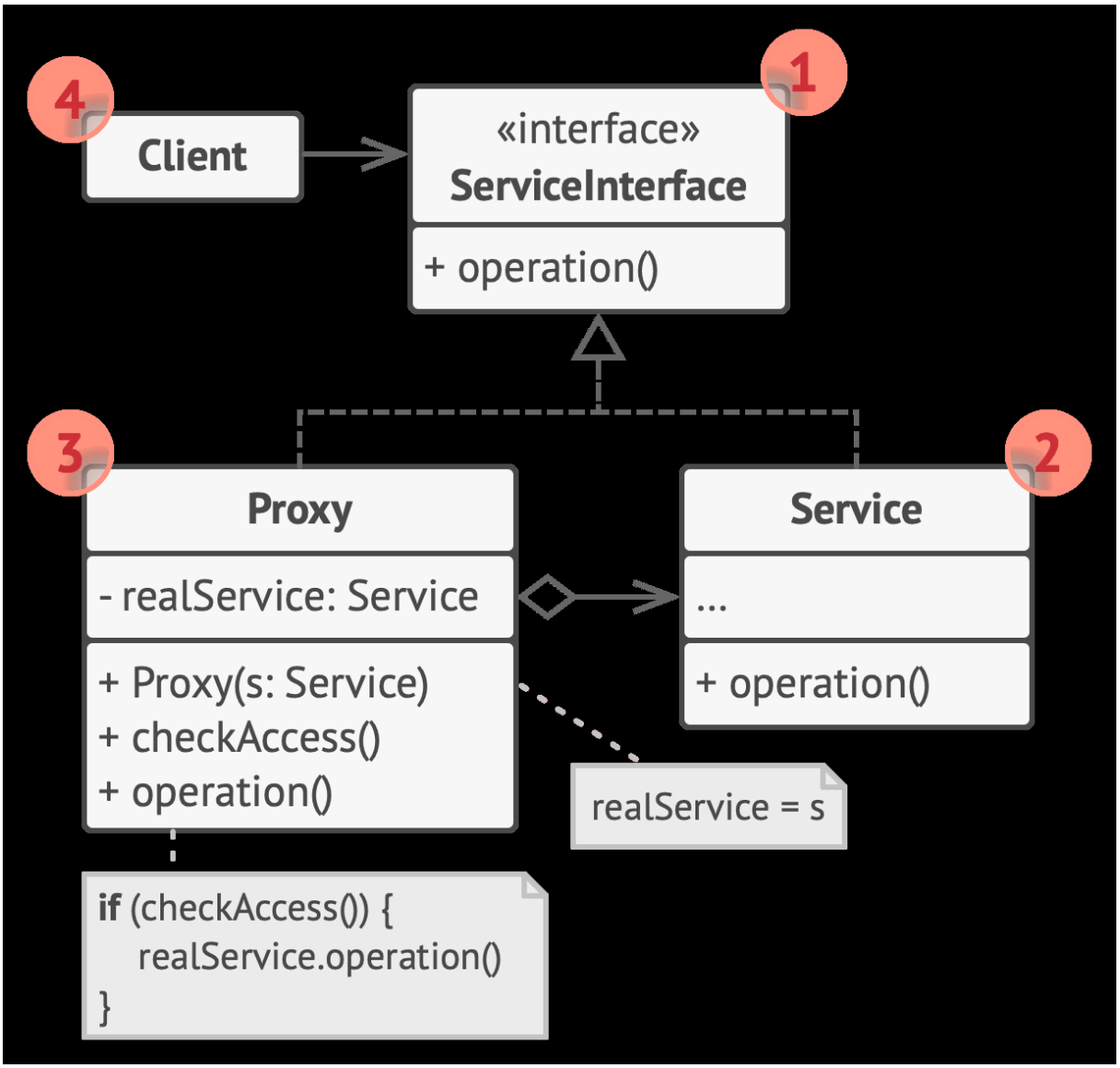
Decorator


Client
a = new ConcComponent()
b = new ConcDecorator1(a)
c = new ConcDecorator2(b)
c.execute()
// Decorator -> Decorator -> Component
5
1
«interface»
Component
+ execute()
2
Concrete
Component
...
+ execute()
3
Base Decorator
- wrappee: Component
+ BaseDecorator(c: Component)
+ execute()
wrappee = c
wrappee.execute()
4
Concrete
Decorators
...
+ execute()
+ extra()
super::execute()
extra()

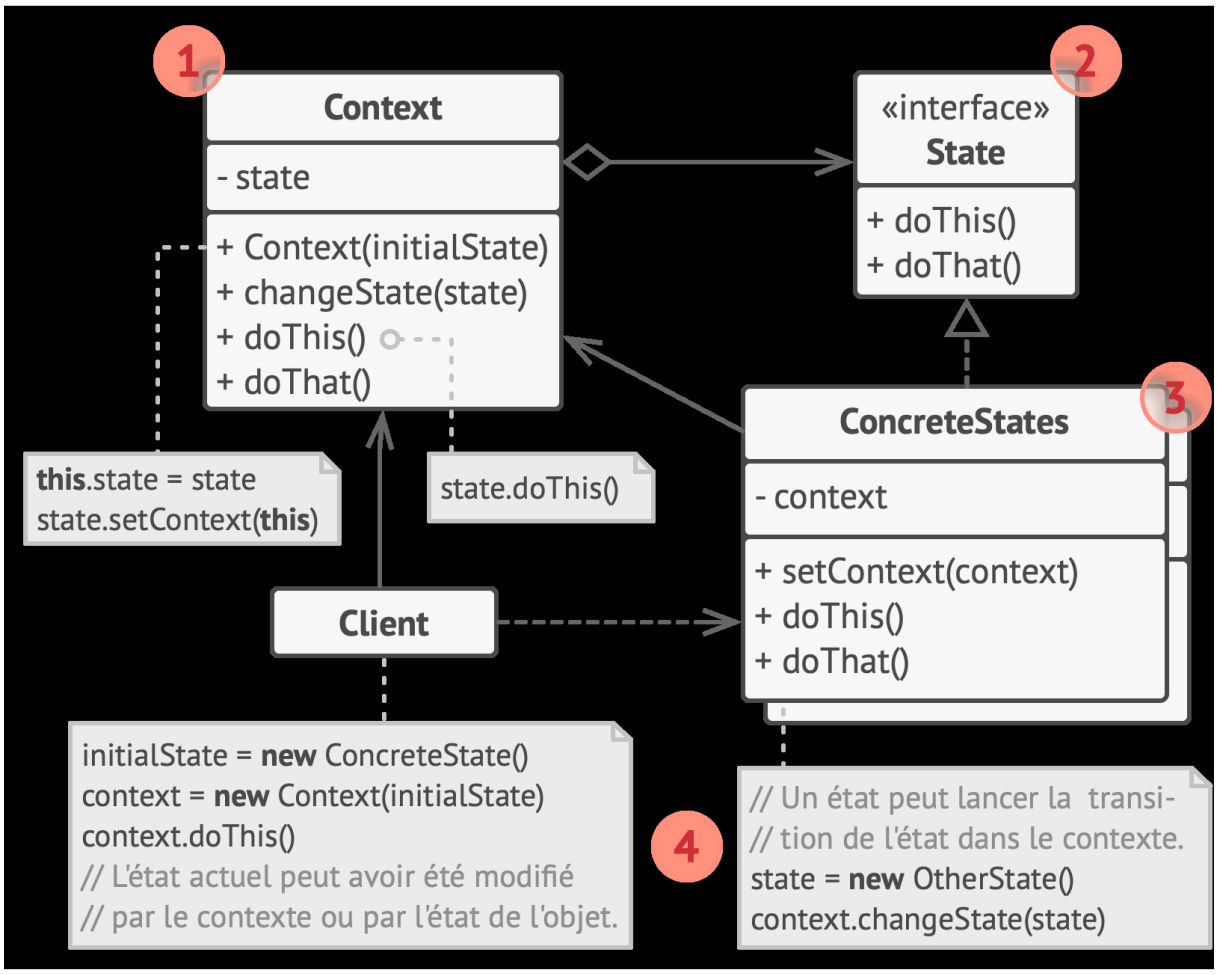
Observer


1
Publisher
- subscribers: Subscriber[]
- mainState
+ subscribe(s: Subscriber)
+ unsubscribe(s: Subscriber)
+ notifySubscribers()
+ mainBusinessLogic()
2
foreach (s in subscribers)
s.update(this)
mainState = newState
notifySubscribers()
3
«interface»
Subscriber
+ update(context)
4
Concrete
Subscribers
...
+ update(context)
5
s = new ConcreteSubscriber()
publisher.subscribe(s)
6
Client

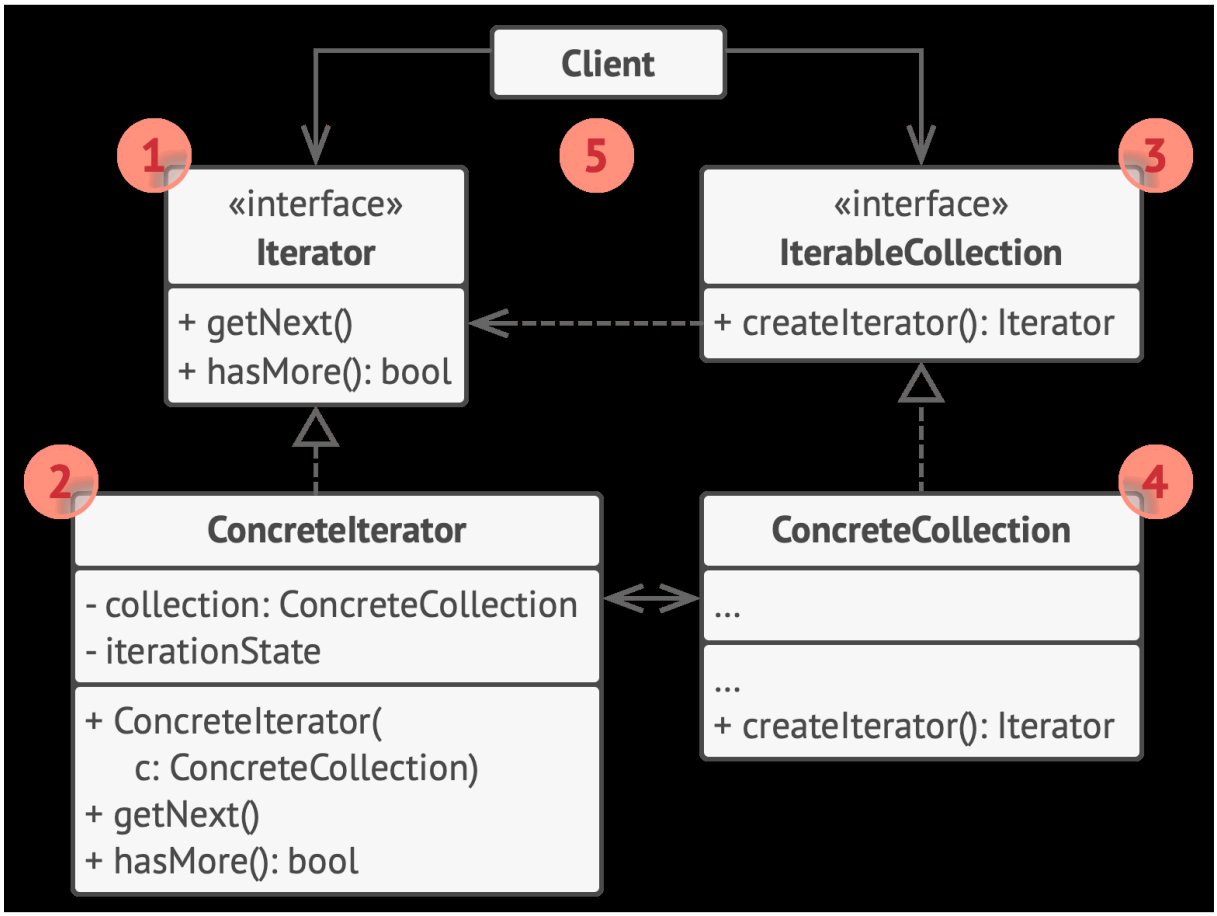
Proxy


4
Client
1
«interface»
ServiceInterface
+ operation()
3
Proxy
- realService: Service
+ Proxy(s: Service)
+ checkAccess()
+ operation()
2
Service
...
+ operation()
realService = s
if (checkAccess()) {
realService.operation()
}

State


1
Context
- state
+ Context(initialState)
+ changeState(state)
+ doThis()
+ doThat()
2
«interface»
State
+ doThis()
+ doThat()
3
ConcreteStates
- context
+ setContext(context)
+ doThis()
+ doThat()
this.state = state
state.setContext(this)
state.doThis()
Client
initialState = new ConcreteState()
context = new Context(initialState)
context.doThis()
// L'état actuel peut avoir été modifié
// par le contexte ou par l'état de l'objet.
4
// Un état peut lancer la transi-
// tion de l'état dans le contexte.
state = new OtherState()
context.changeState(state)

Iterator


Client
1
5
3
«interface»
Iterator
+ getNext()
+ hasMore(): bool
«interface»
IterableCollection
+ createIterator(): Iterator
2
4
ConcreteIterator
- collection: ConcreteCollection
- iterationState
+ ConcreteIterator(
c: ConcreteCollection)
+ getNext()
+ hasMore(): bool
ConcreteCollection
...
...
+ createIterator(): Iterator