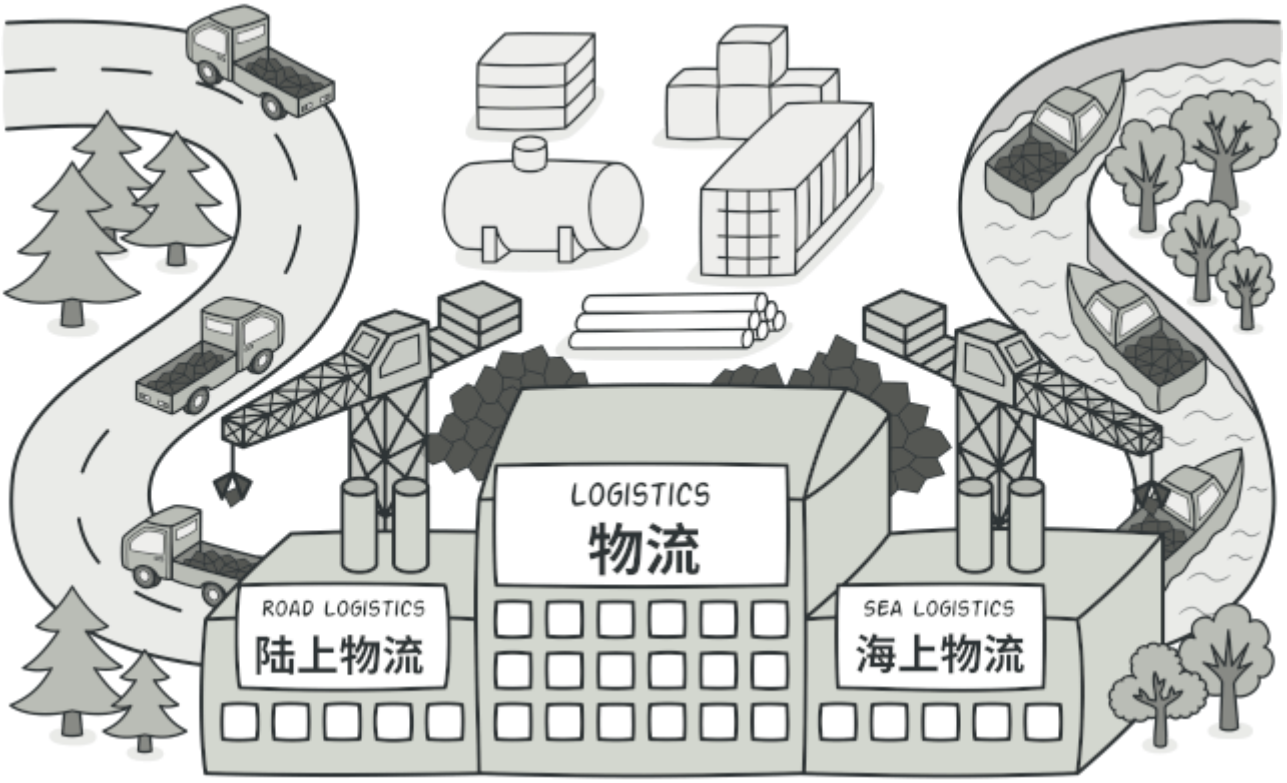
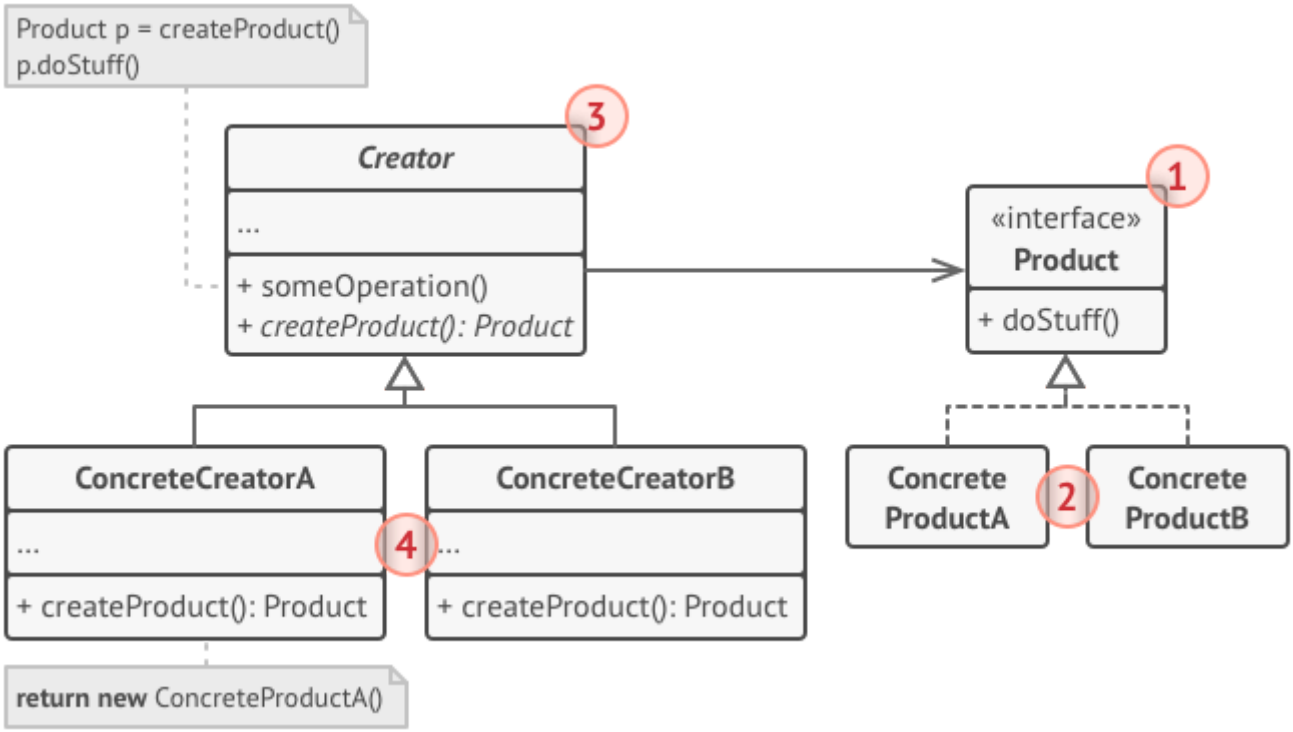
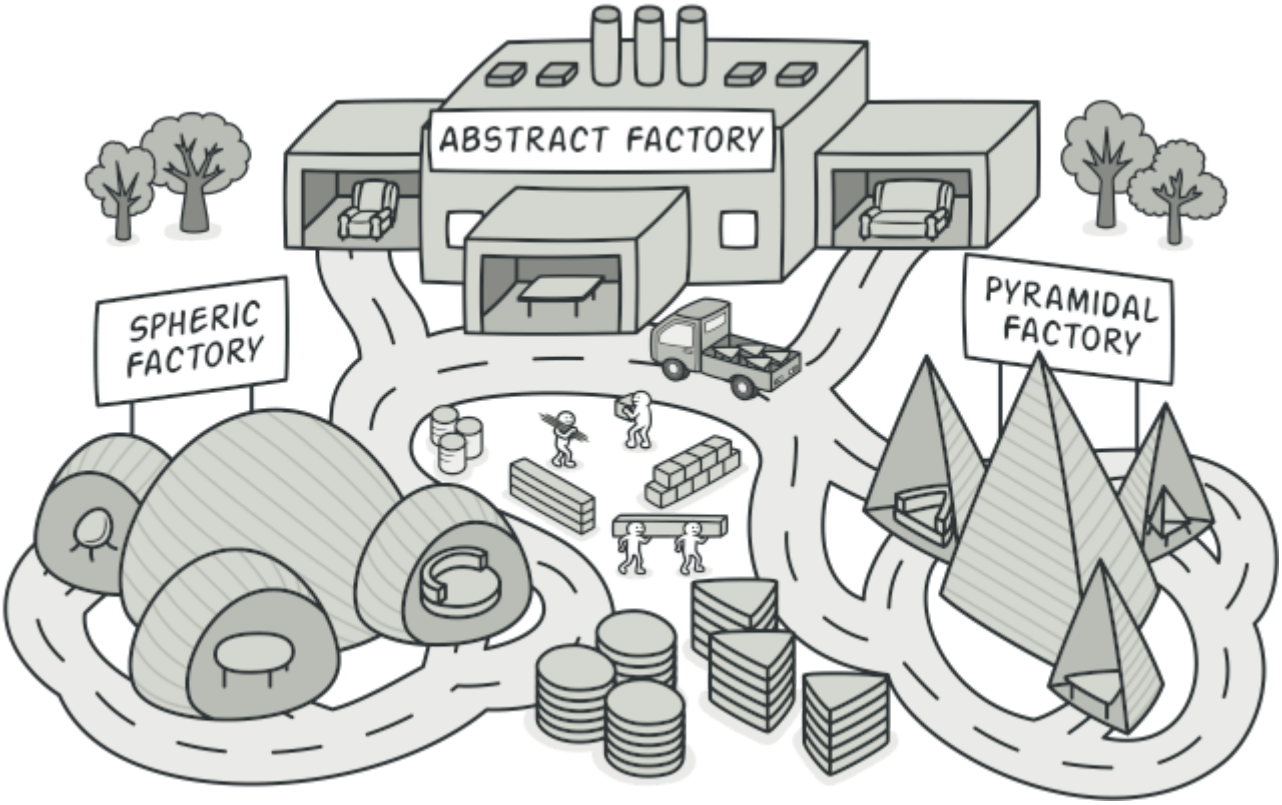
1. Creational Patterns

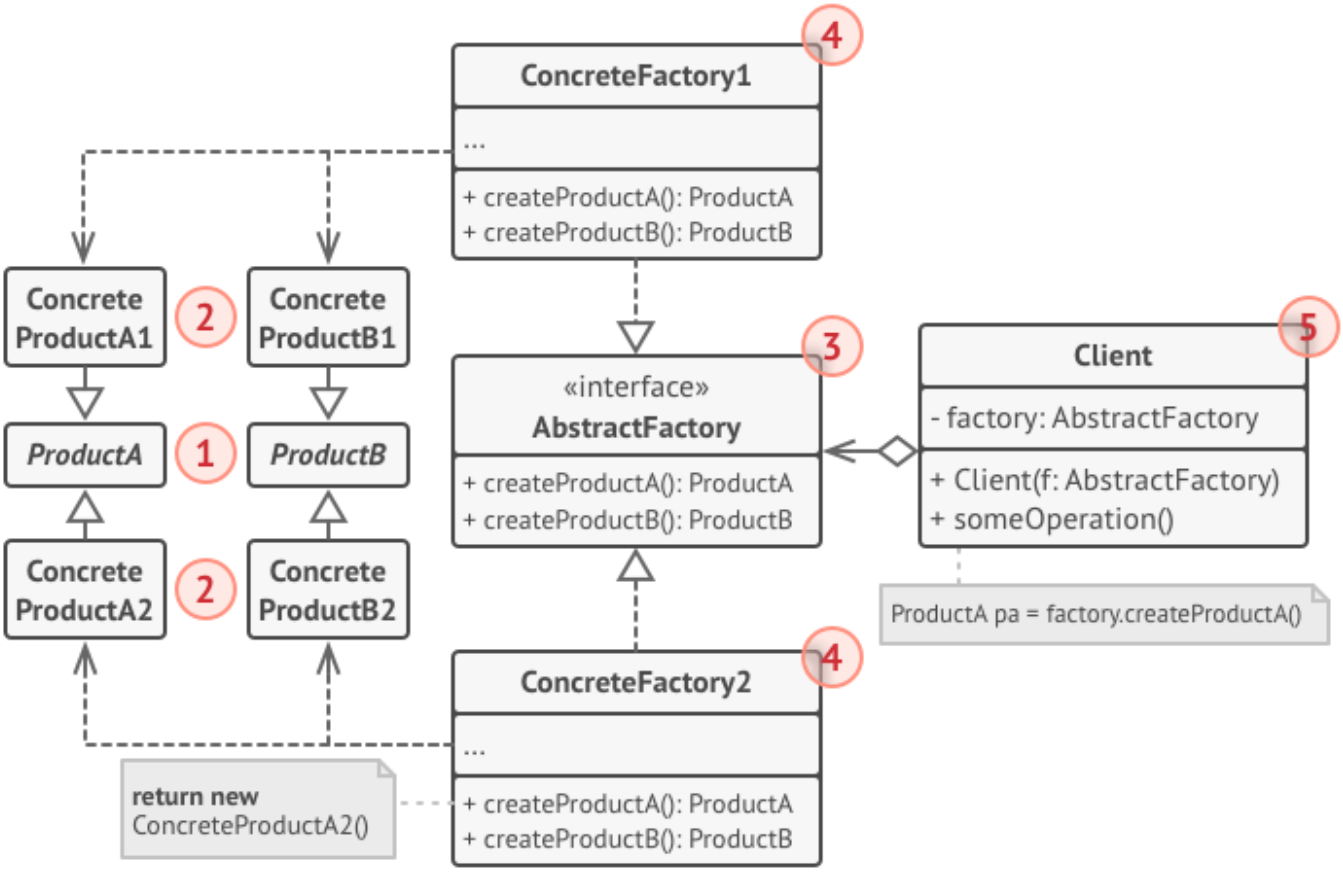
1.1 Factory Method



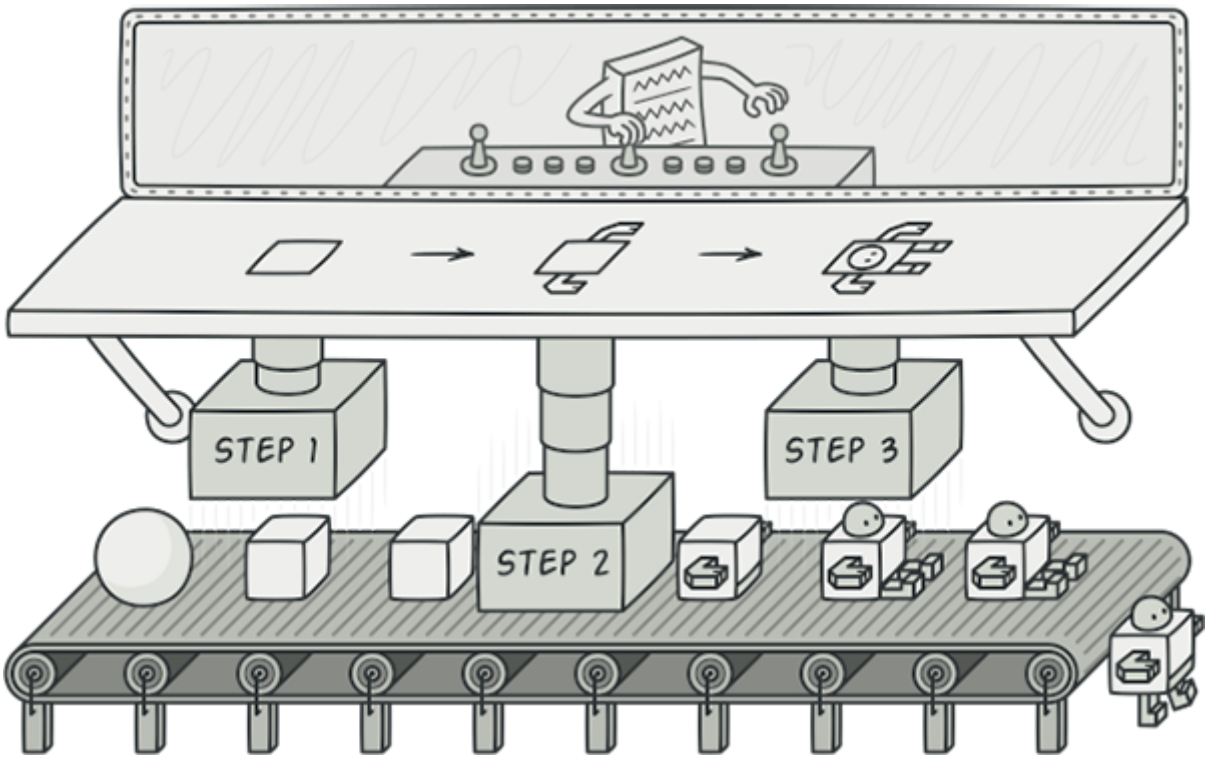


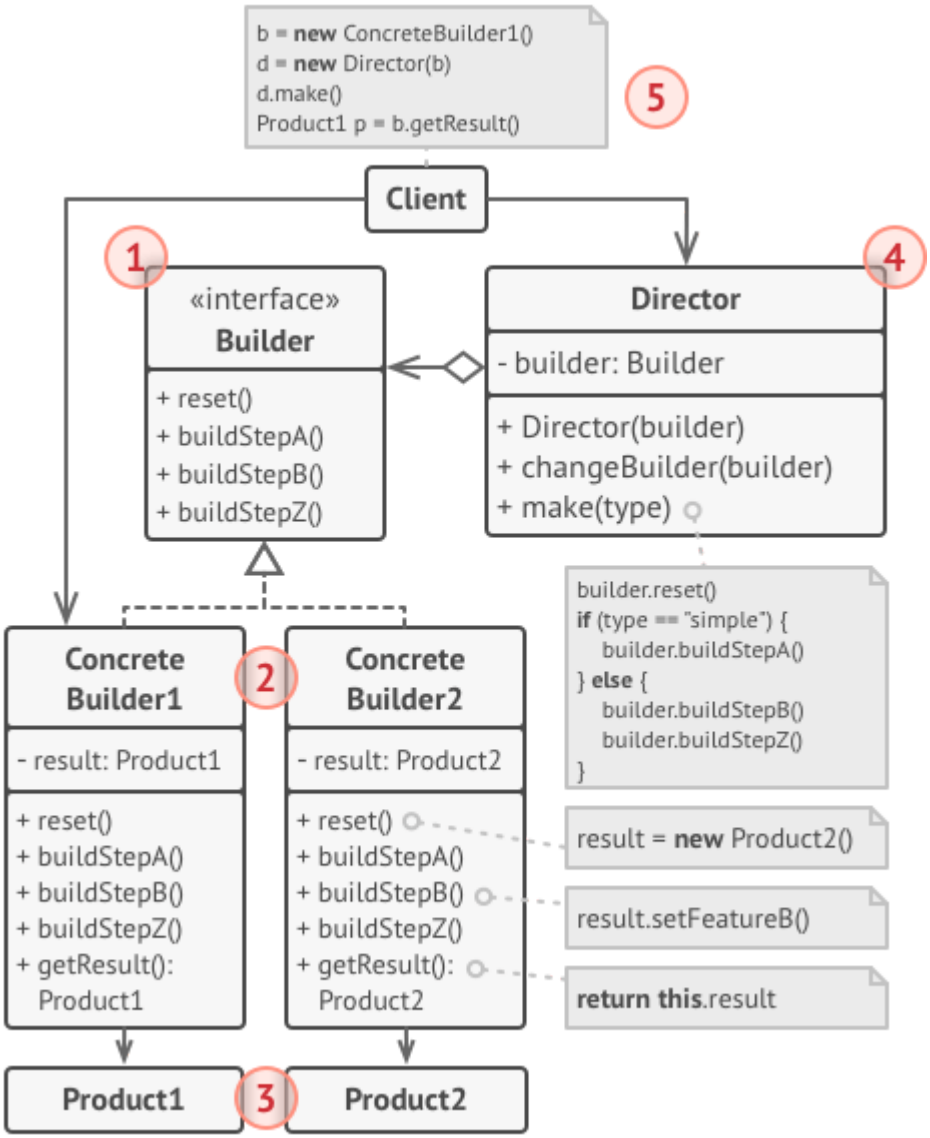
1.2 Abstract Factory



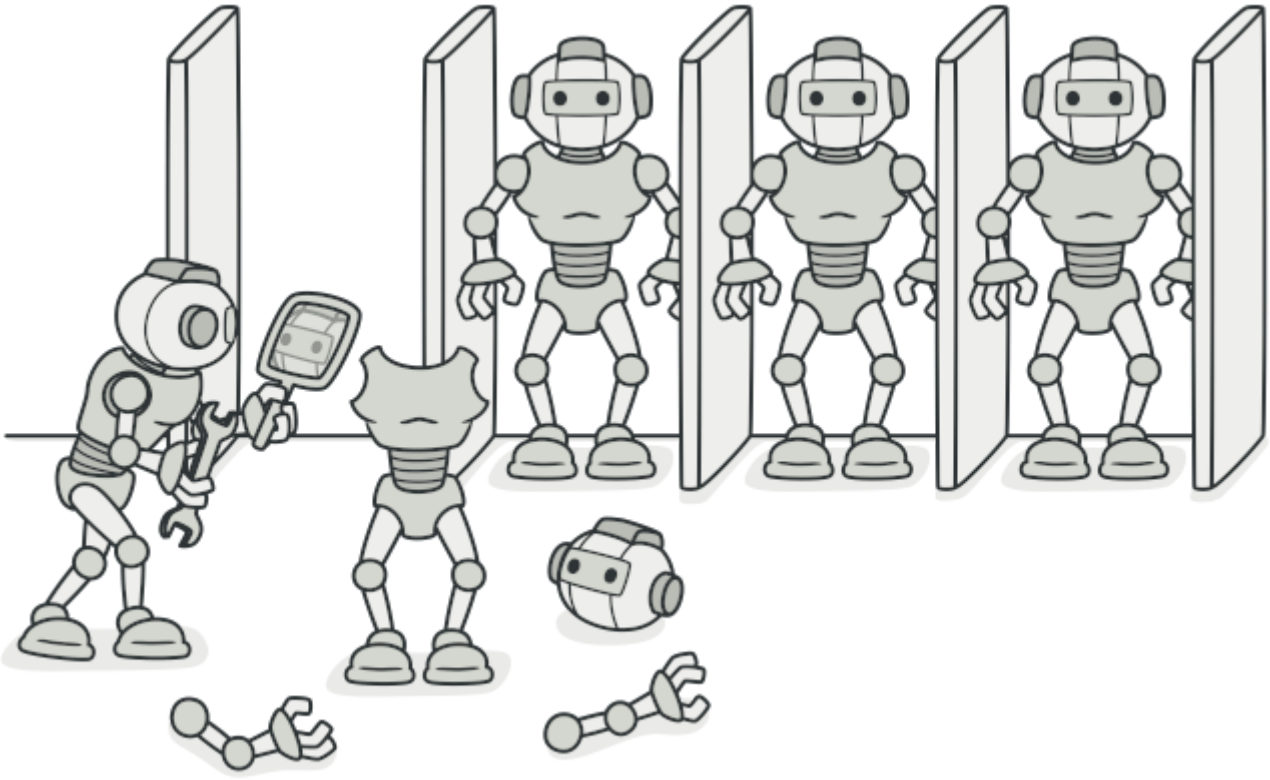


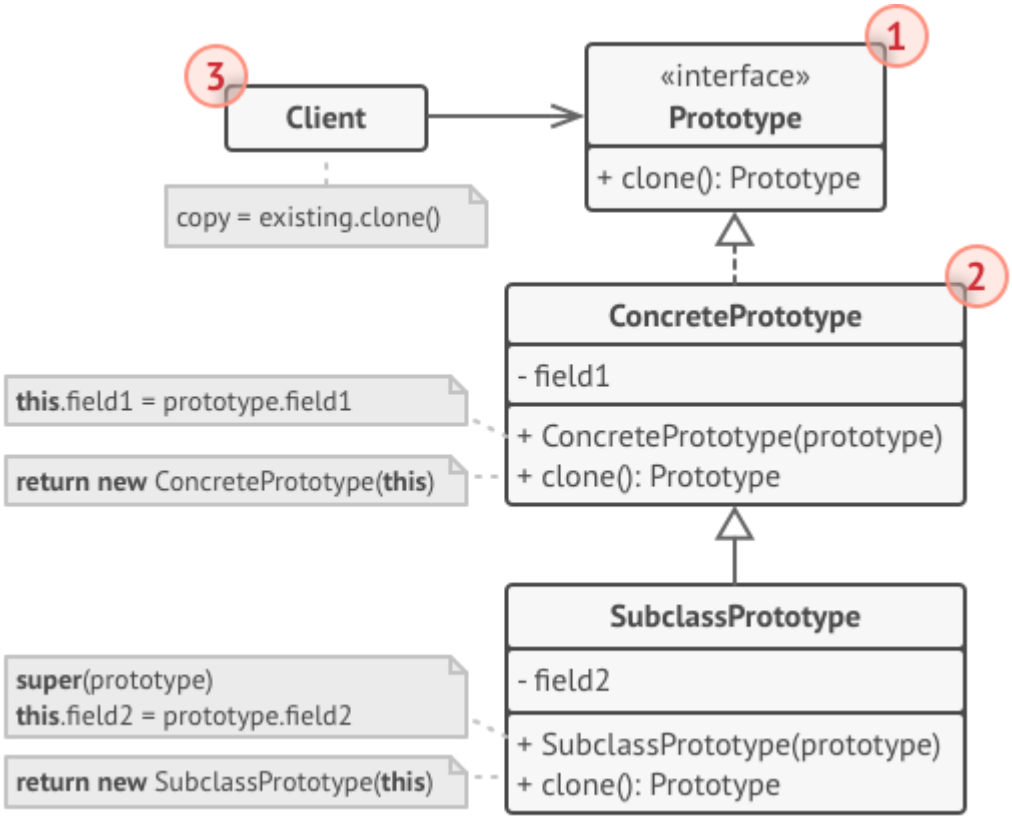
1.3 Builder


STEP 1
STEP 2
STEP 3

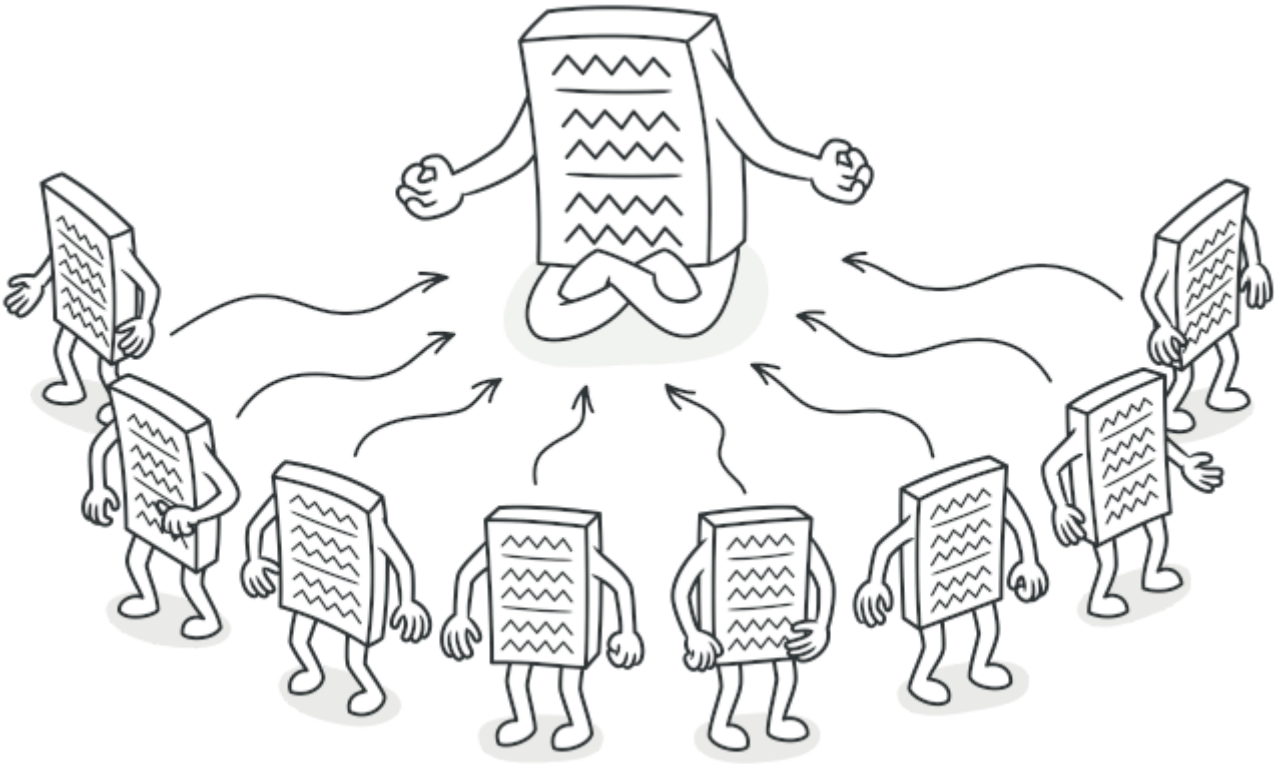

b = new ConcreteBuilder1()
d = new Director(b)
d.make()
Product1 p = b.getResult()
5
Client
1
«interface»
Builder
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
Director
- builder: Builder
+ Director(builder)
+ changeBuilder(builder)
+ make(type)
4
builder.reset()
if (type == "simple") {
builder.buildStepA()
} else {
builder.buildStepB()
builder.buildStepZ()
}
Concrete
Builder1
- result: Product1
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
+ getResult():
Product1
2
Concrete
Builder2
- result: Product2
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
+ getResult():
Product2
result = new Product2()
result.setFeatureB()
return this.result
Product1
3
Product2

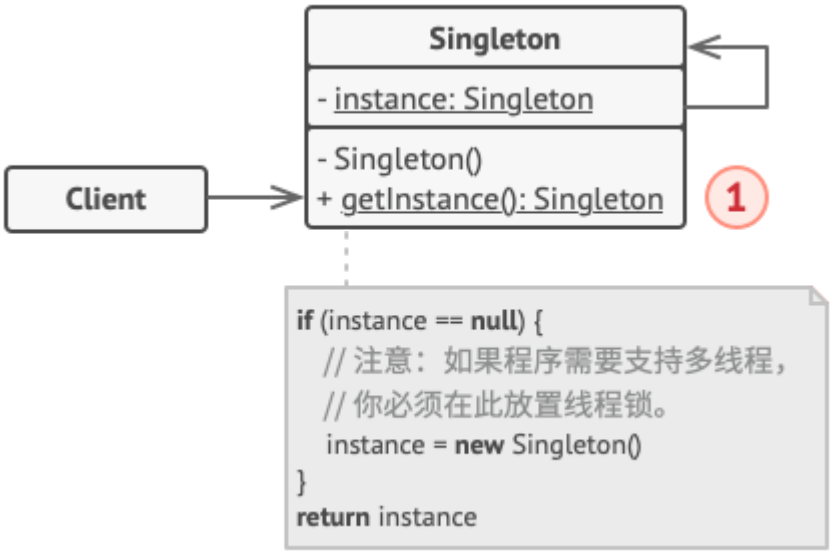
1.4 Prototype





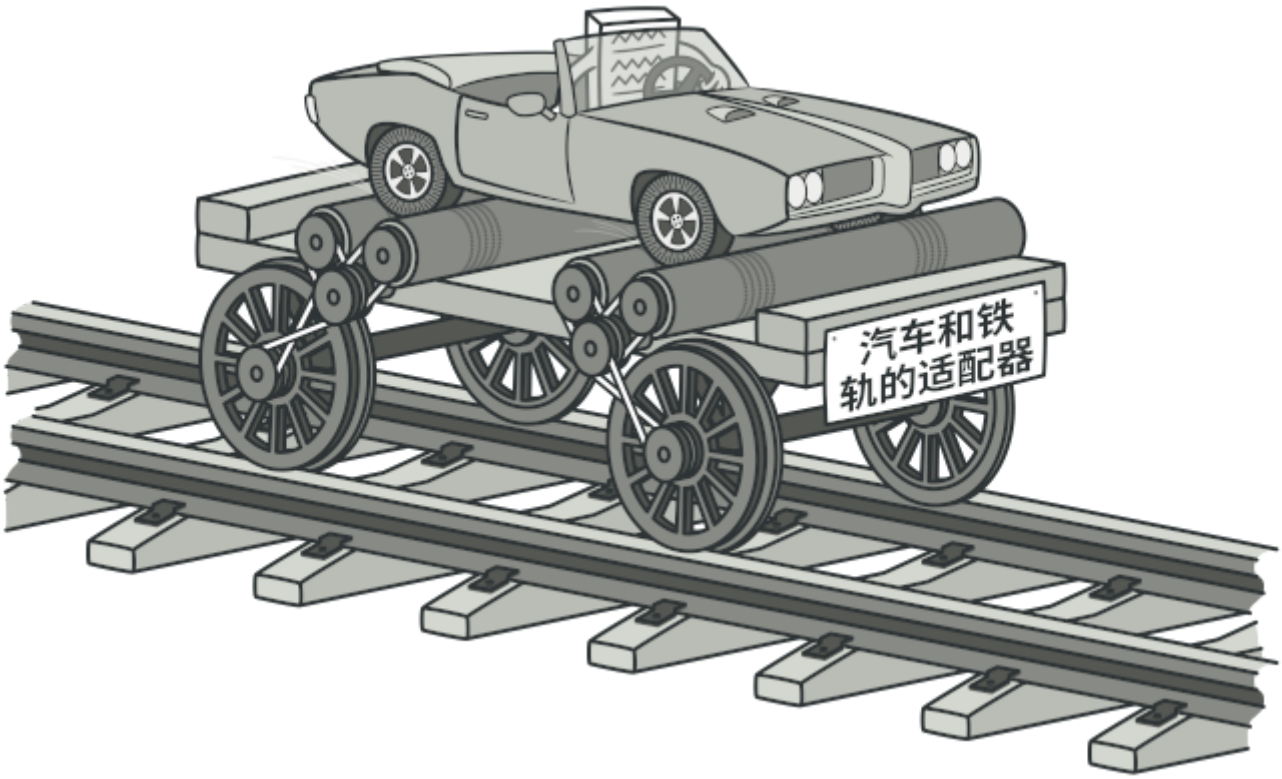
1.5 Singleton

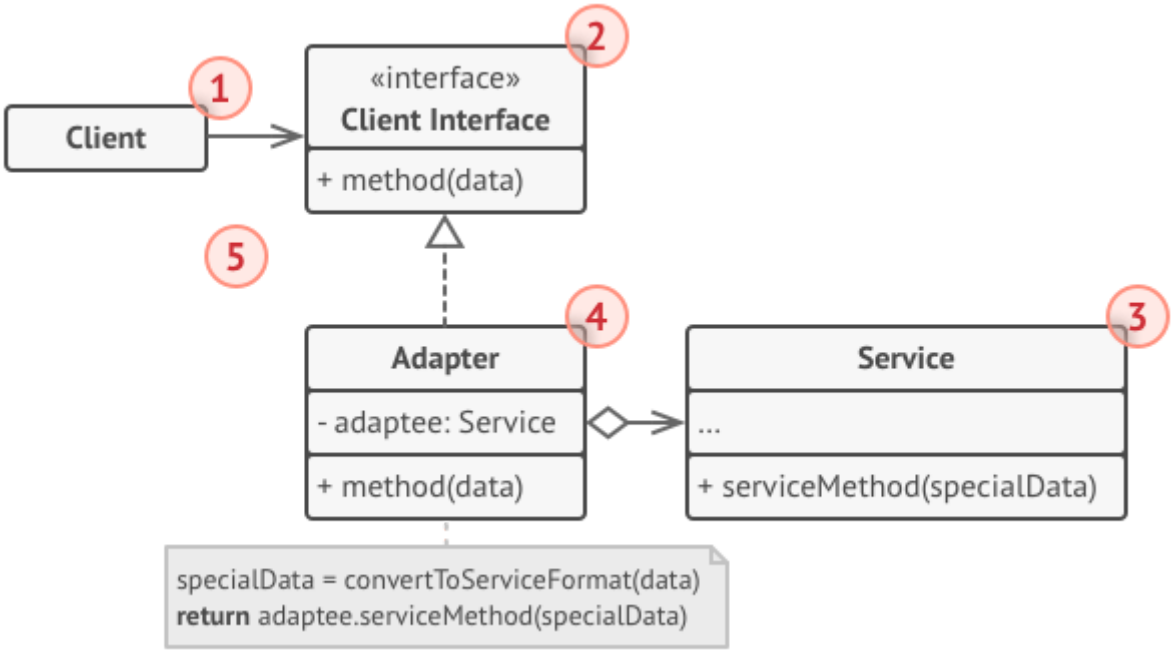




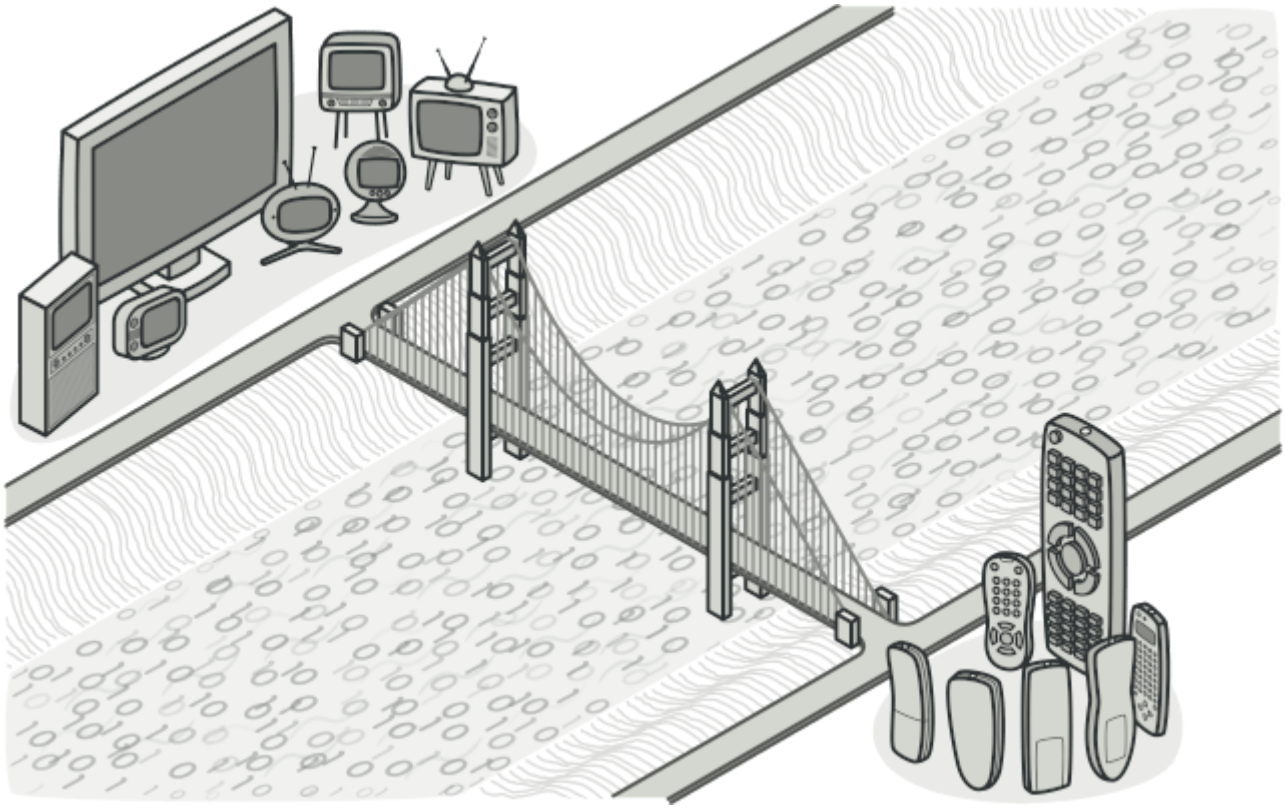
2. Structural Patterns

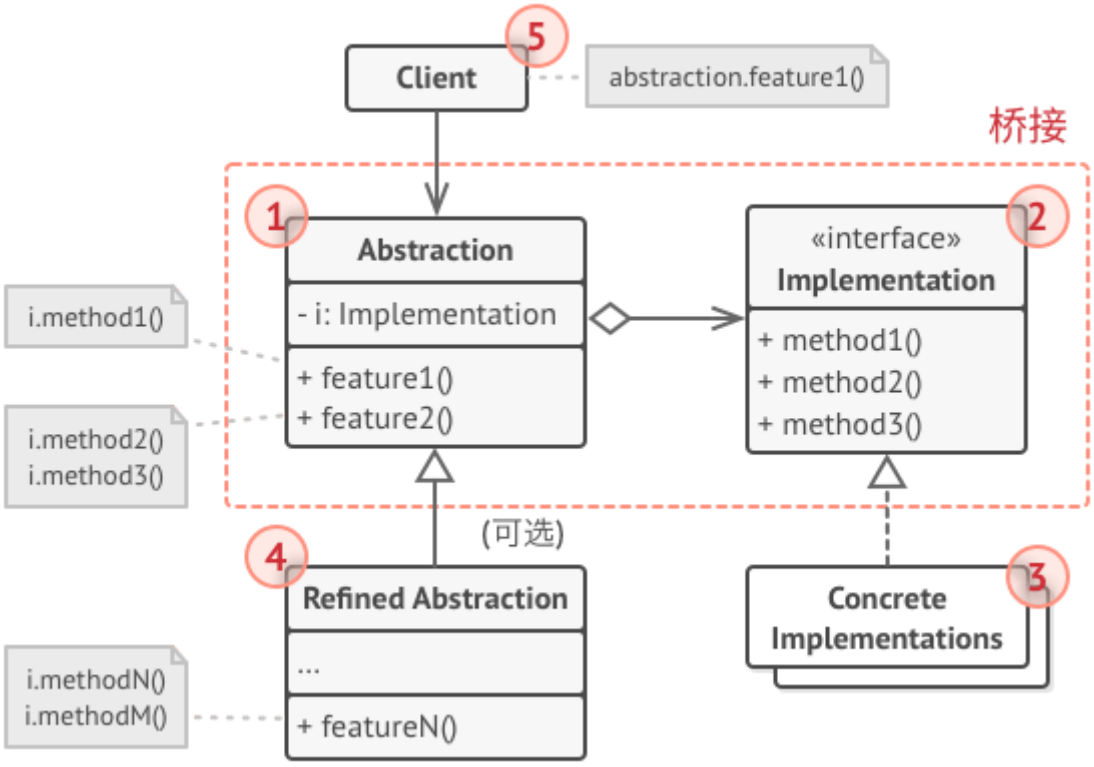
2.1 Adapter



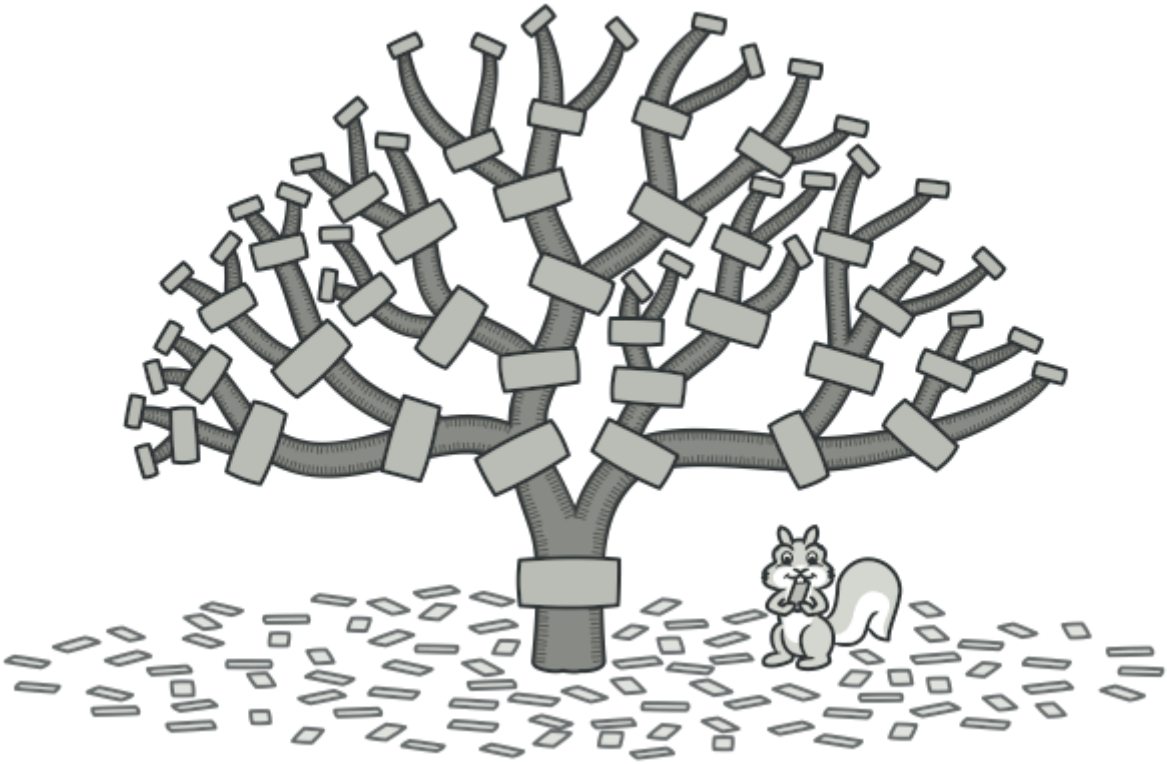


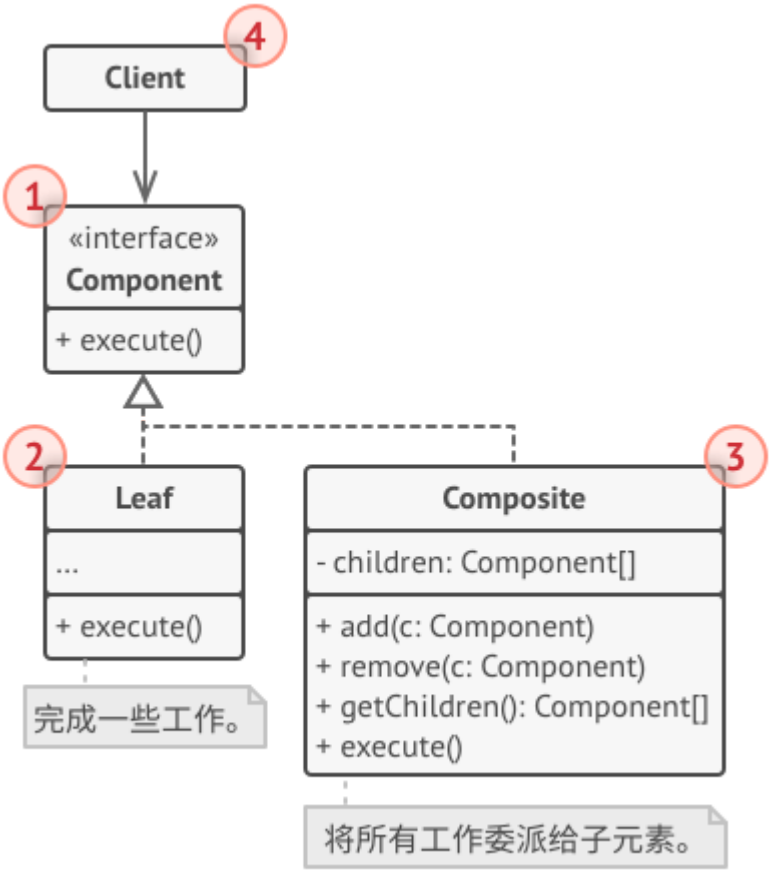
2.2 Bridge



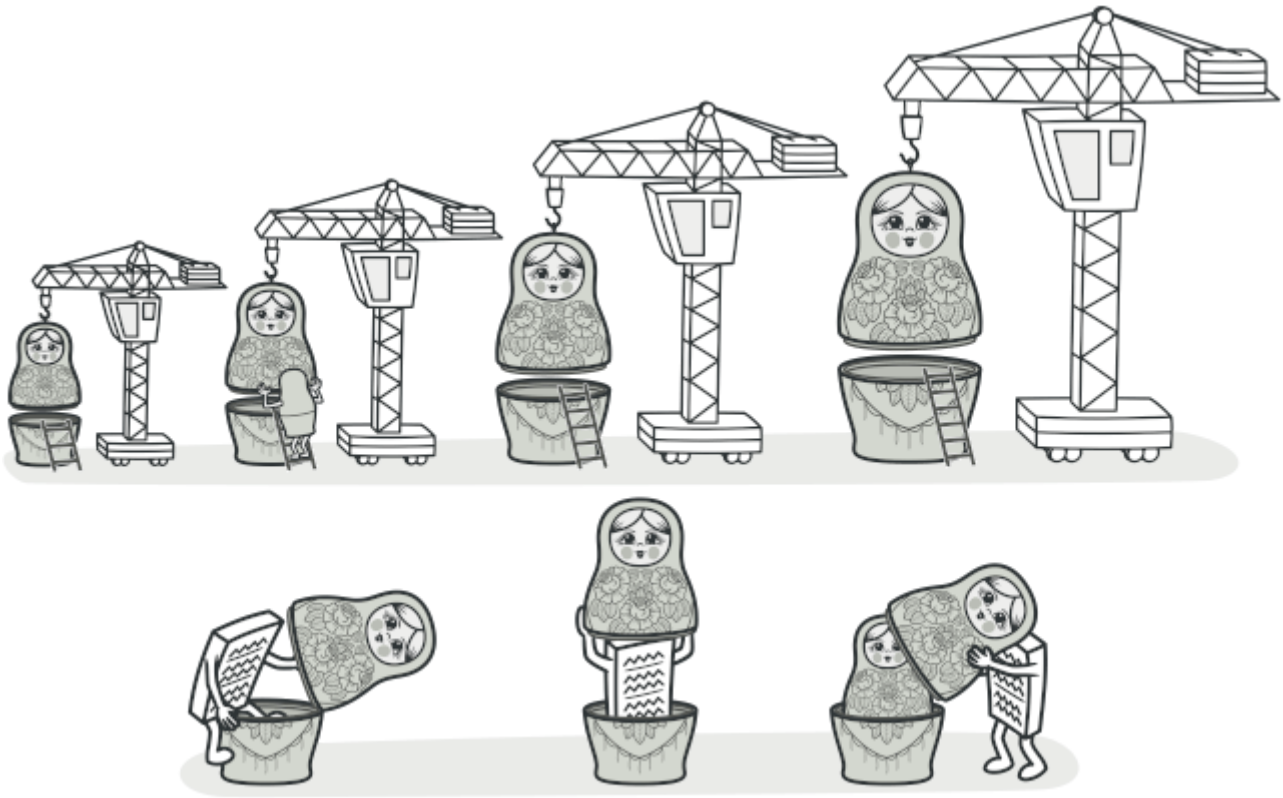

5
Client
abstraction.feature1()
桥接
1
Abstraction
- i: Implementation
+ feature1()
+ feature2()
2
«interface»
Implementation
+ method1()
+ method2()
+ method3()
i.method1()
i.method2()
i.method3()
(可选)
4
Refined Abstraction
...
+ featureN()
3
Concrete
Implementations
i.methodN()
i.methodM()

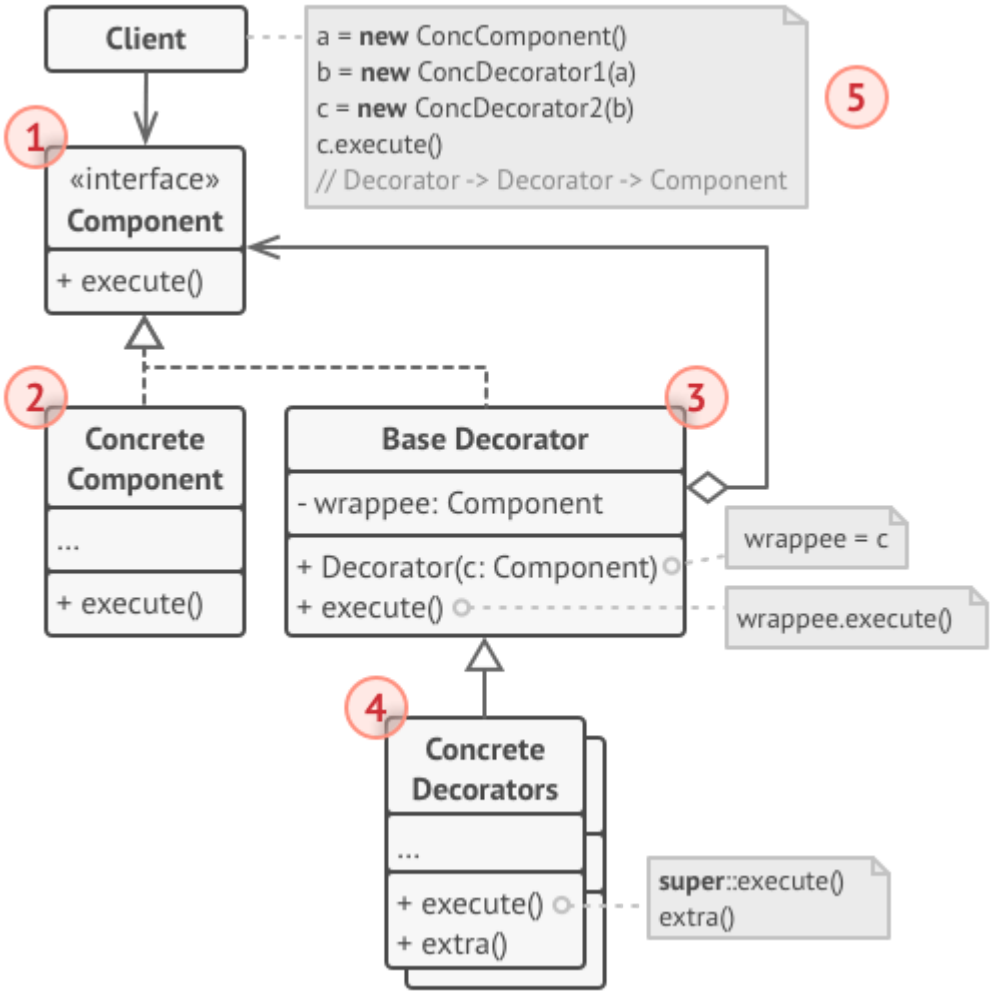
2.3 Composite



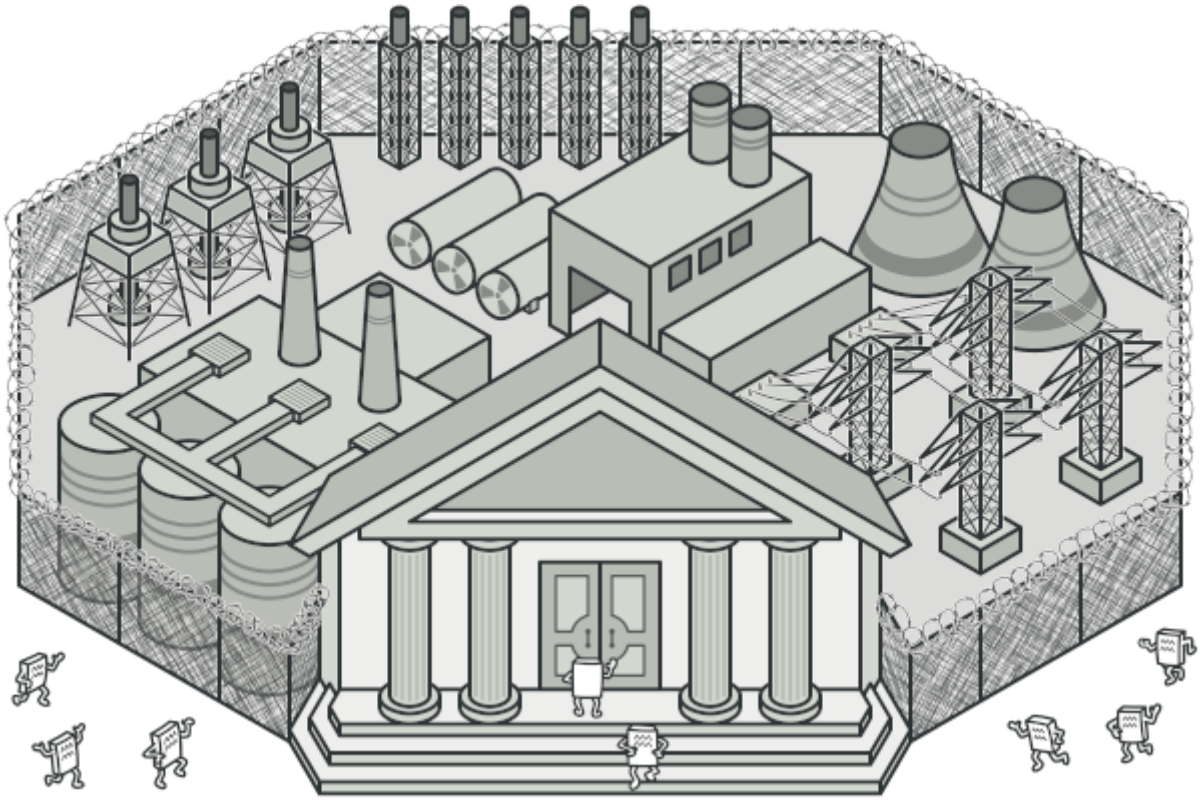


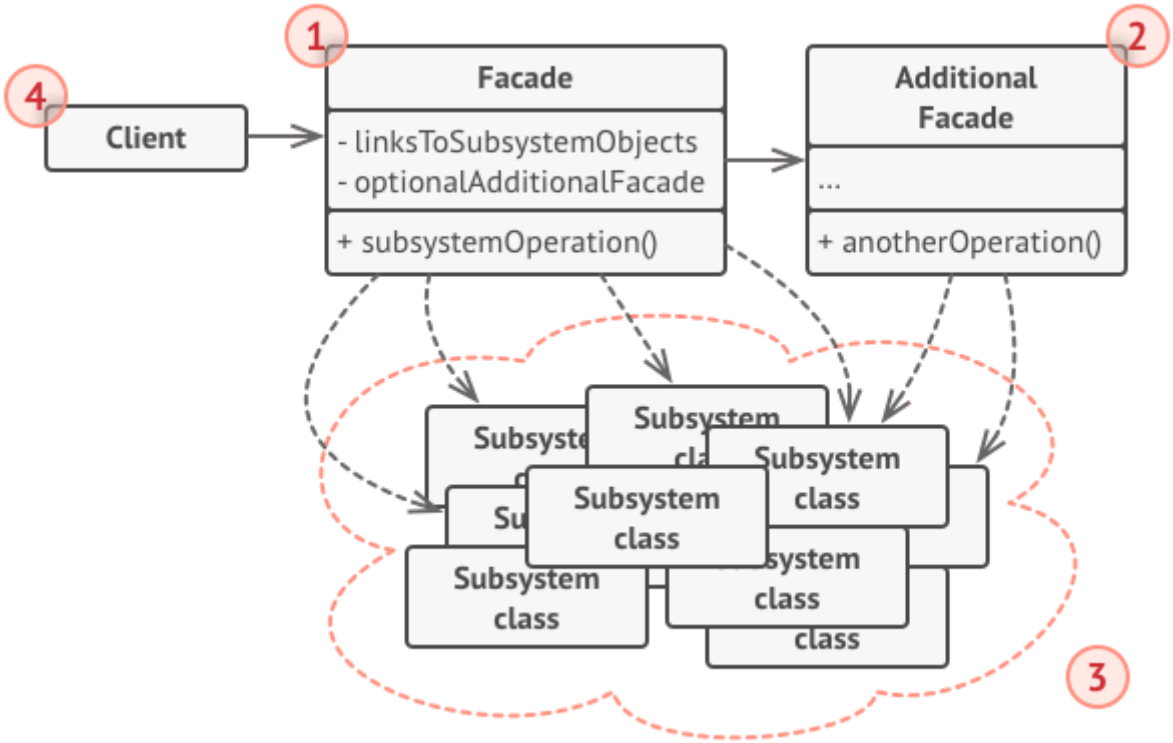
2.4 Decorator



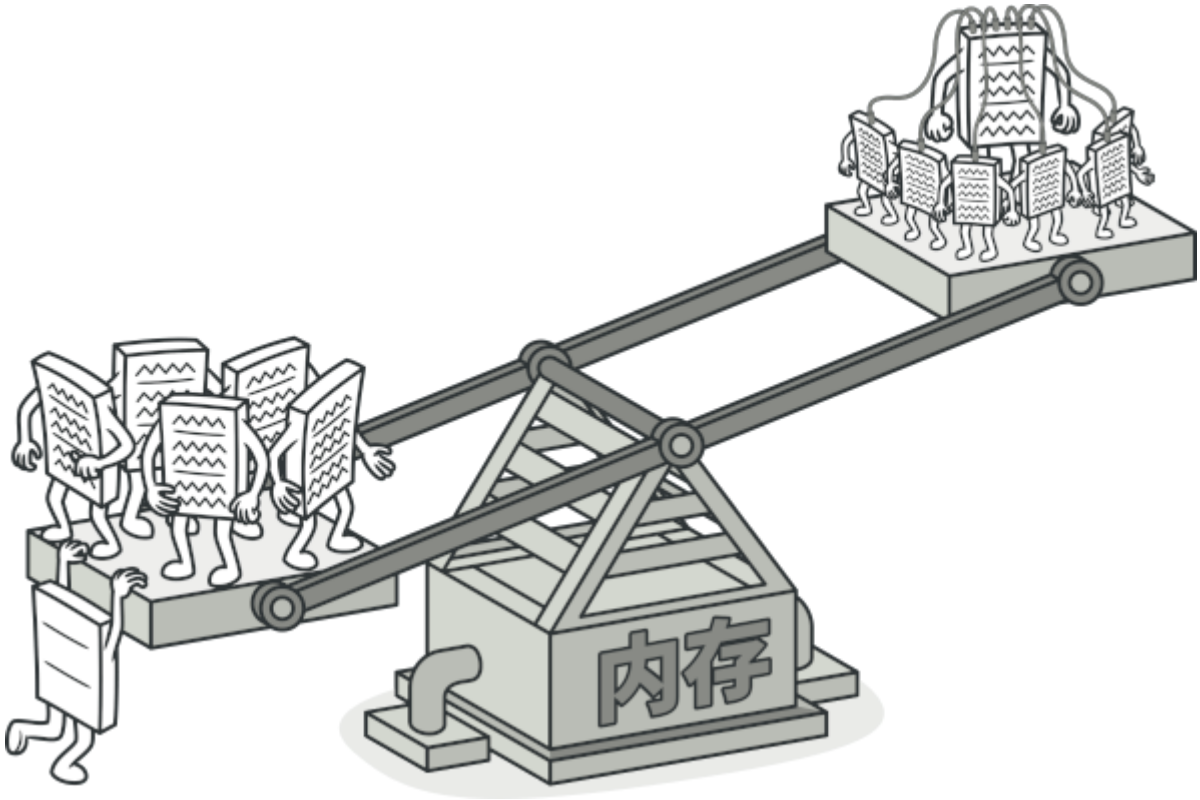


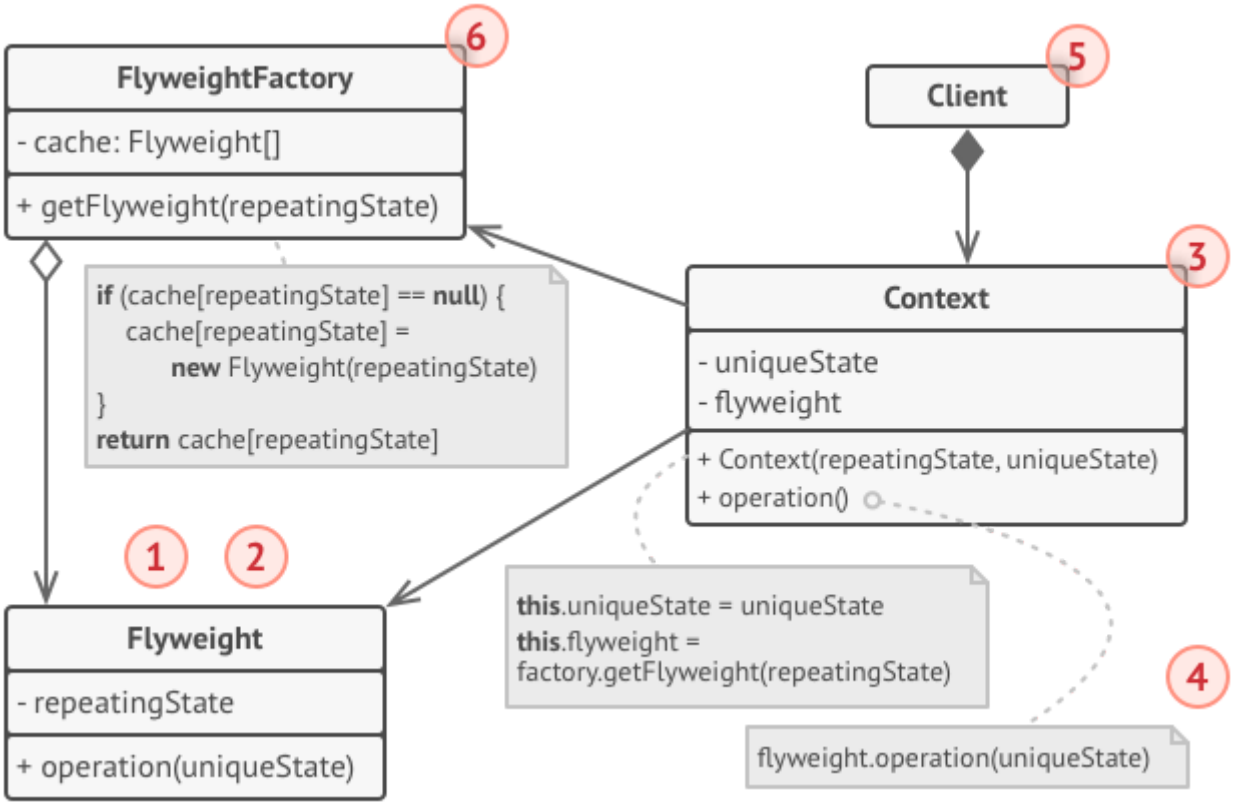
2.5 Facade



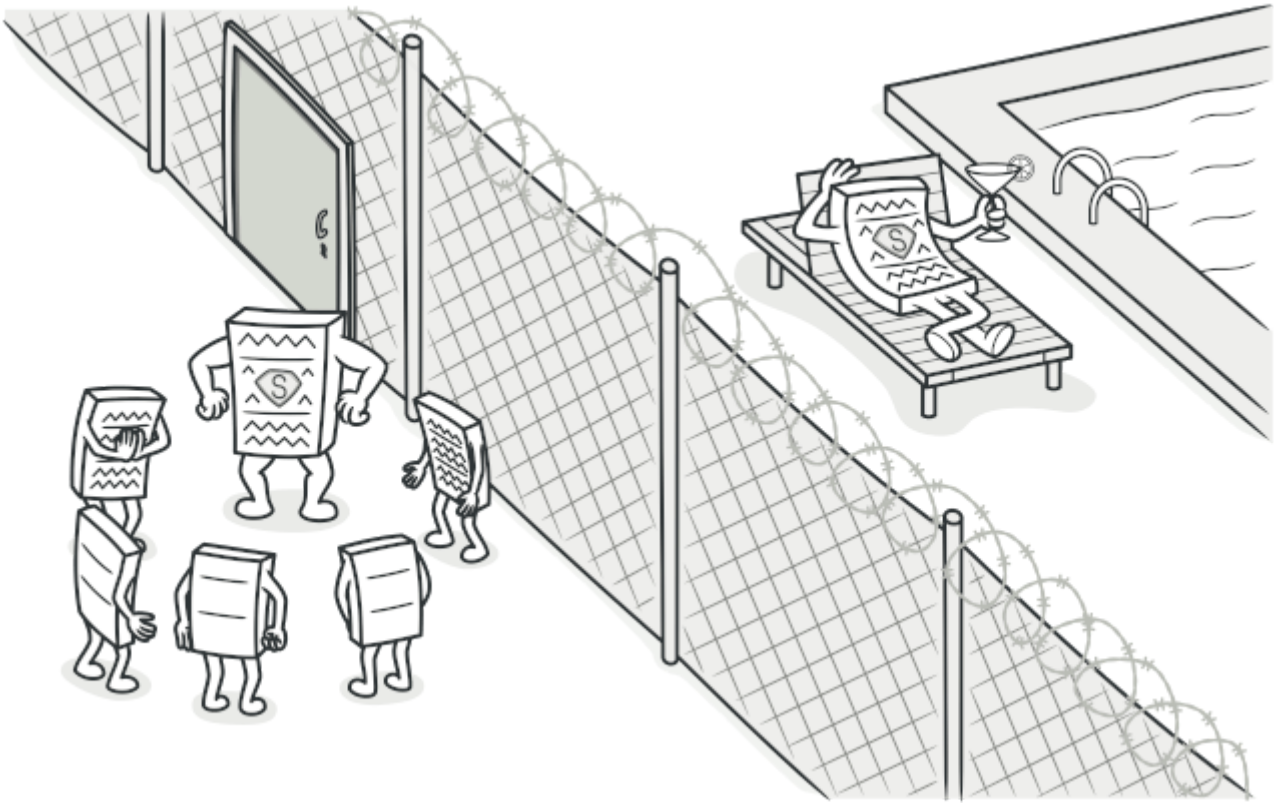

1
Facade
- linksToSubsystemObjects
- optionalAdditionalFacade
+ subsystemOperation()
2
Additional Facade
...
+ anotherOperation()
4
Client
Subsystem class
Subsystem class
Subsystem class
Subsystem class
3

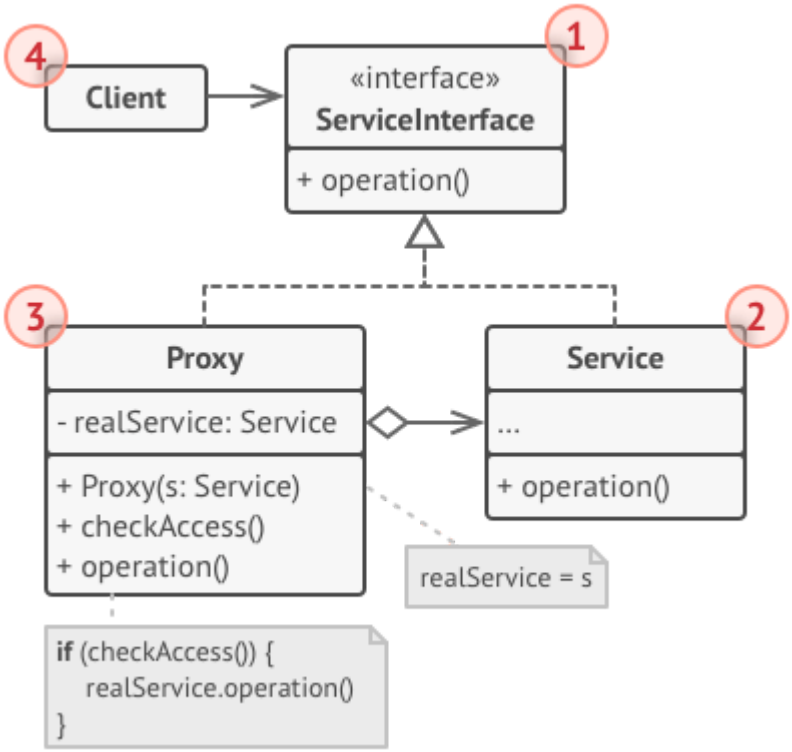
2.6 Flyweight





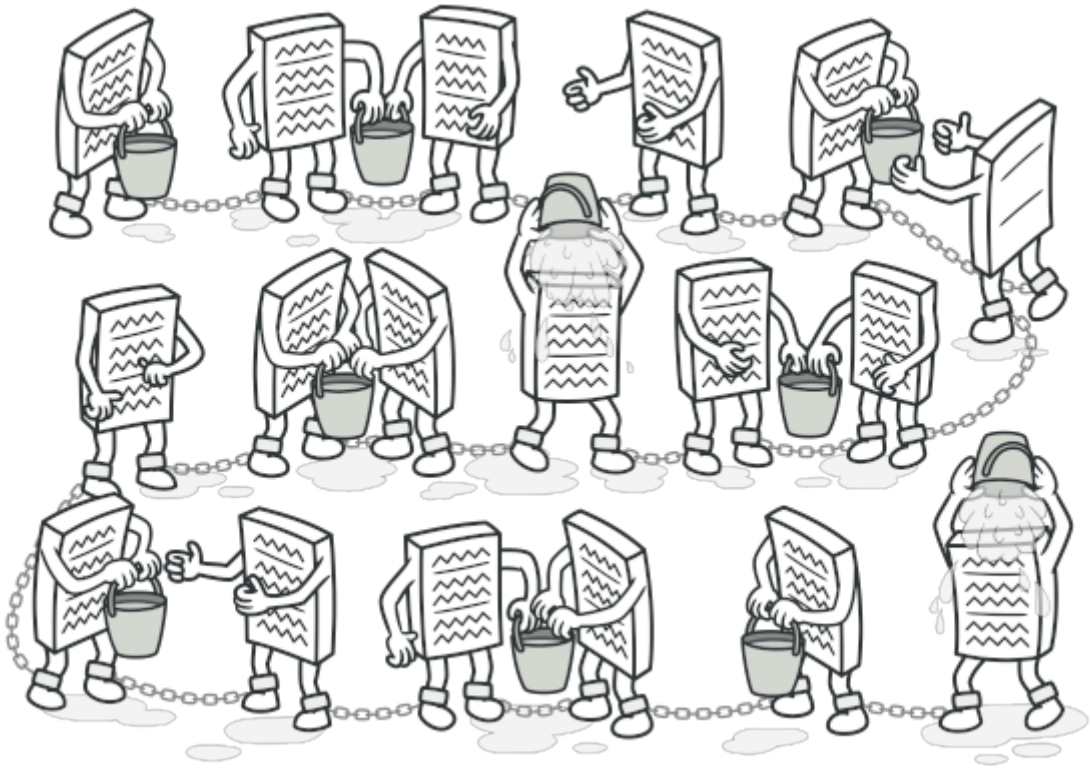
2.7 Proxy

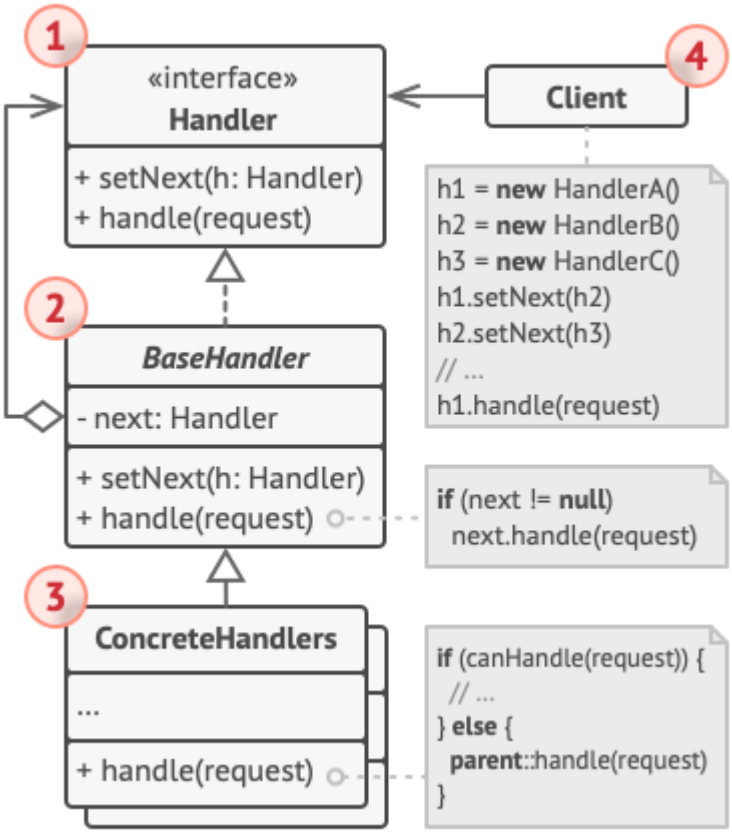



4
Client
1
«interface»
ServiceInterface
+ operation()
3
Proxy
- realService: Service
+ Proxy(s: Service)
+ checkAccess()
+ operation()
2
Service
...
+ operation()
realService = s
if (checkAccess()) {
realService.operation()
}

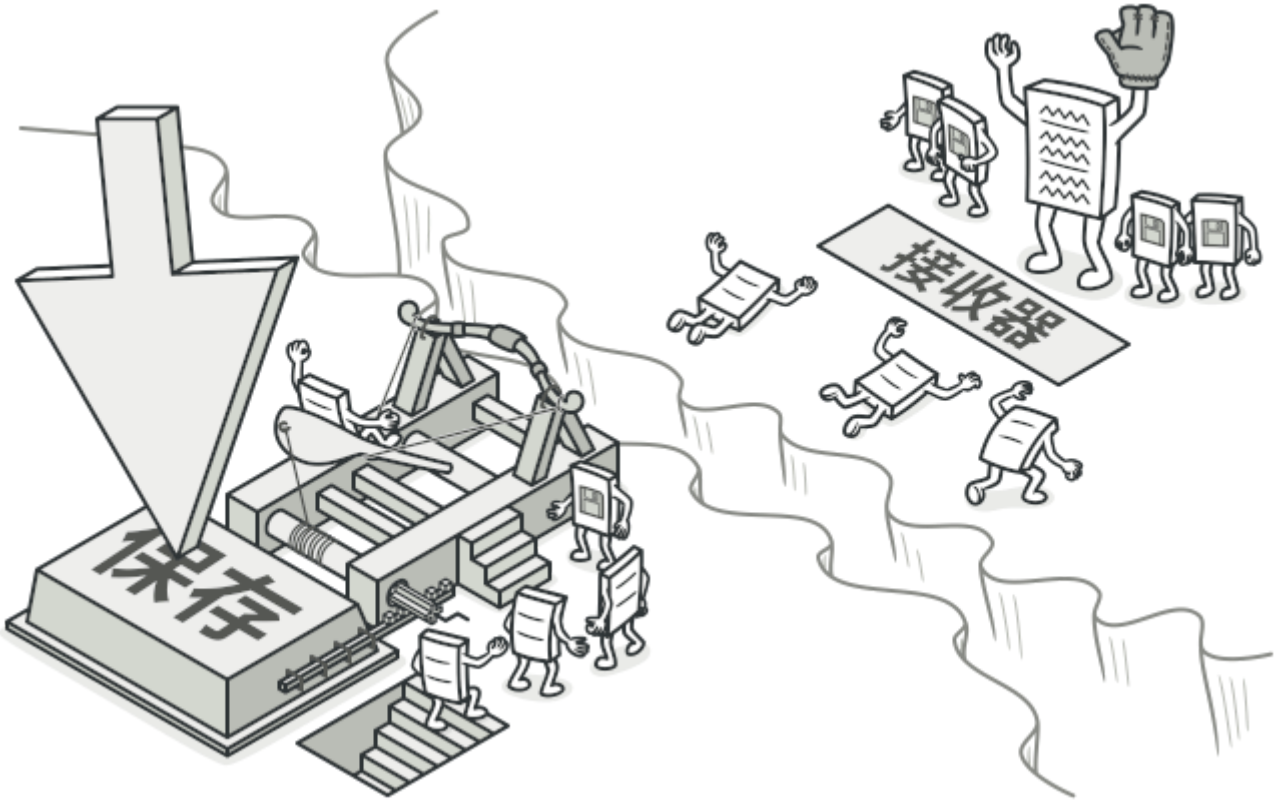
3. Behavioral Patterns

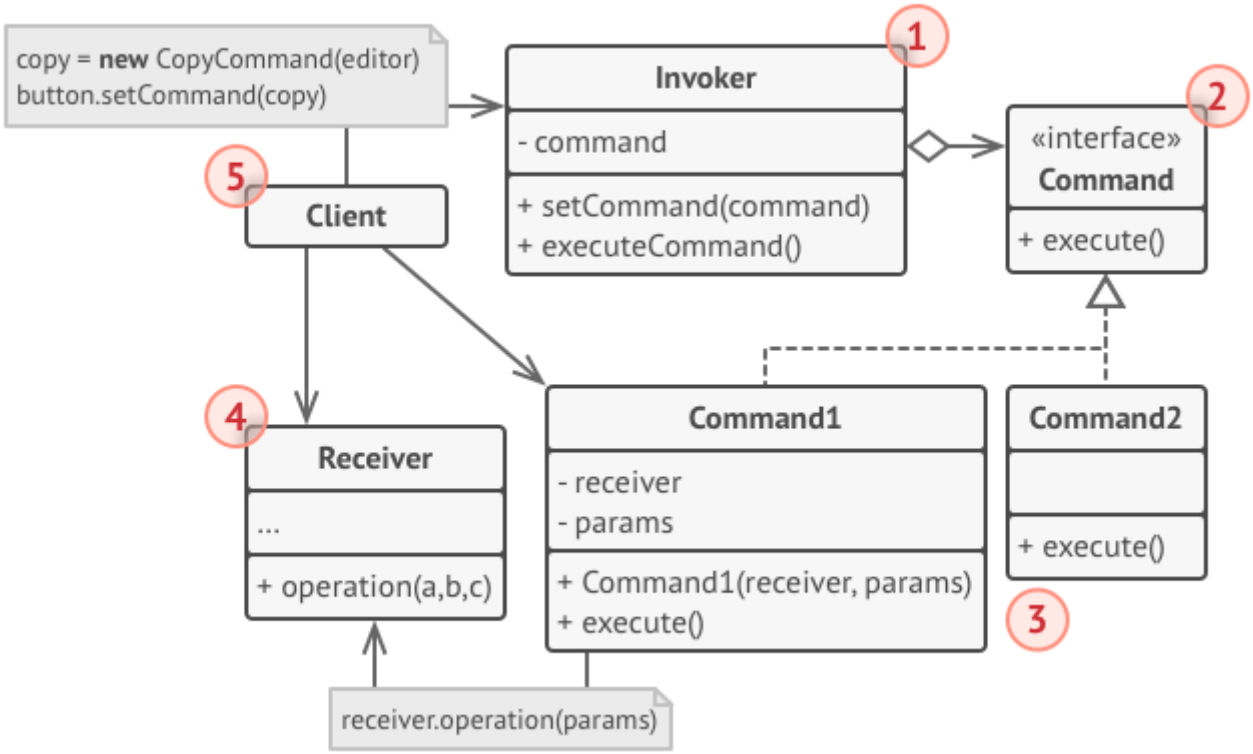
3.1 Chain of Responsibility



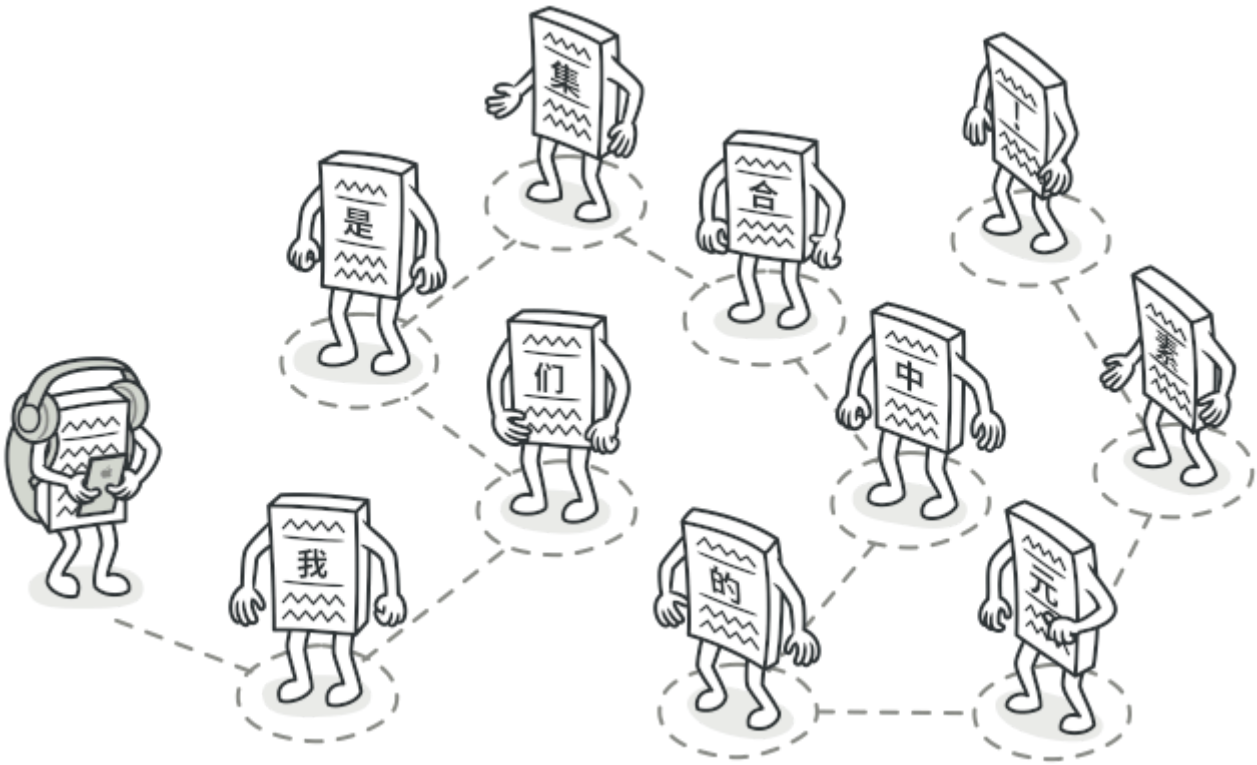


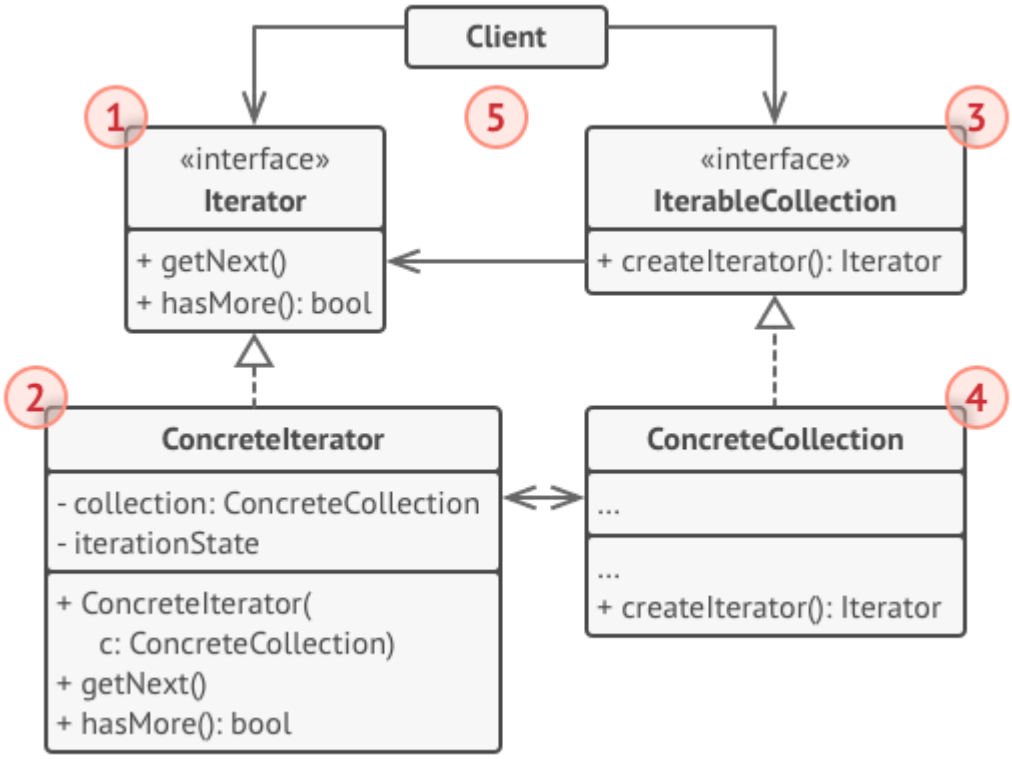
3.2 Command


保存
接收器

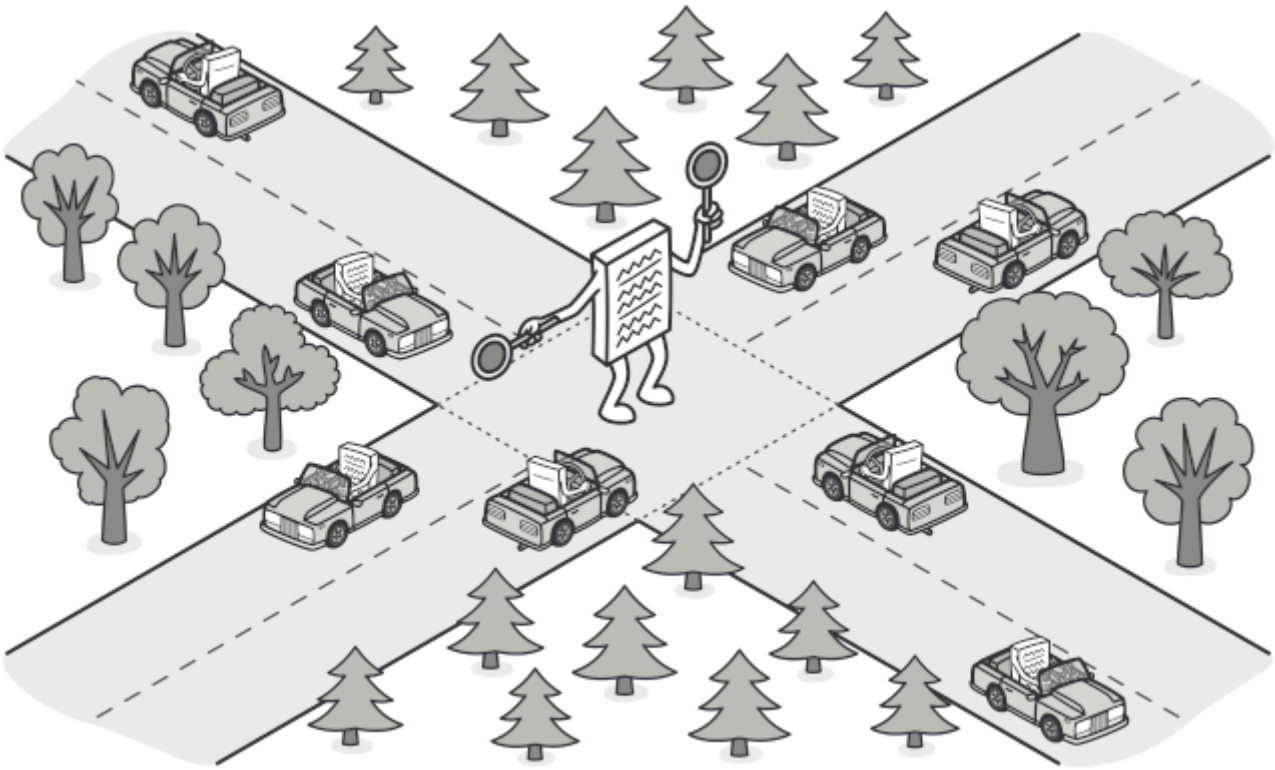

copy = new CopyCommand(editor)
button.setCommand(copy)
Invoker
- command
+ setCommand(command)
+ executeCommand()
«interface»
Command
+ execute()
Client
Receiver
...
+ operation(a,b,c)
Command1
- receiver
- params
+ Command1(receiver, params)
+ execute()
Command2
+ execute()
receiver.operation(params)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

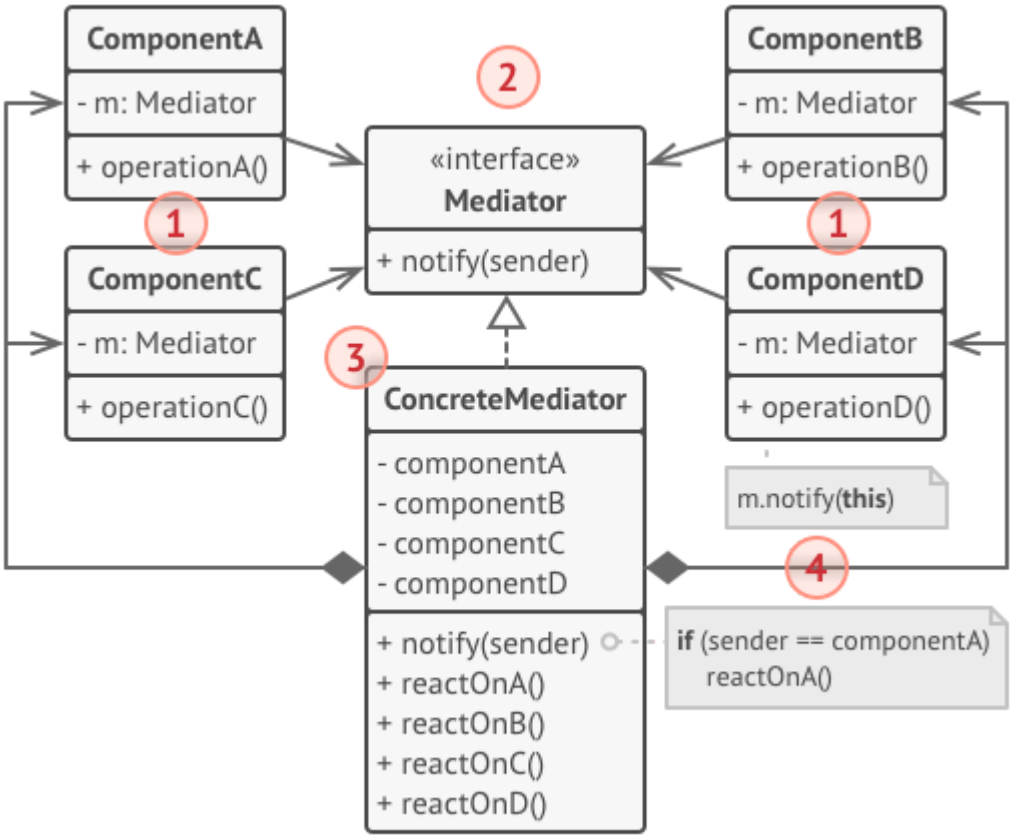
3.3 Iterator


集
合
一
是
们
中
我
的
元


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

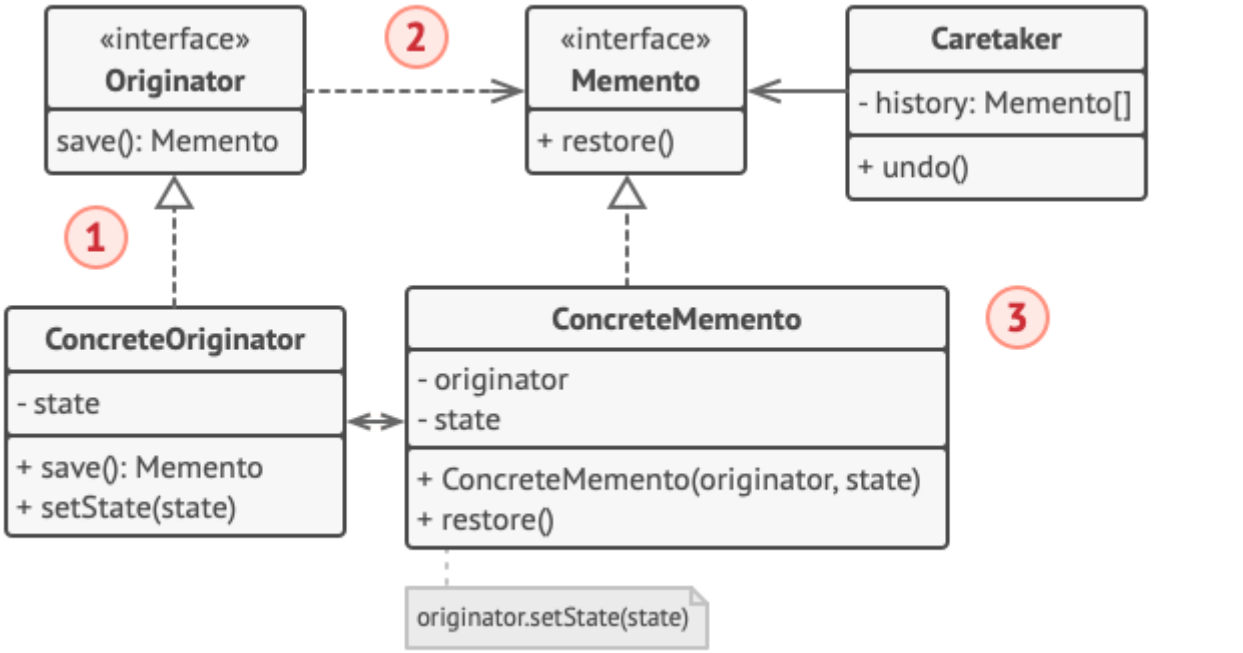
3.4 Mediator



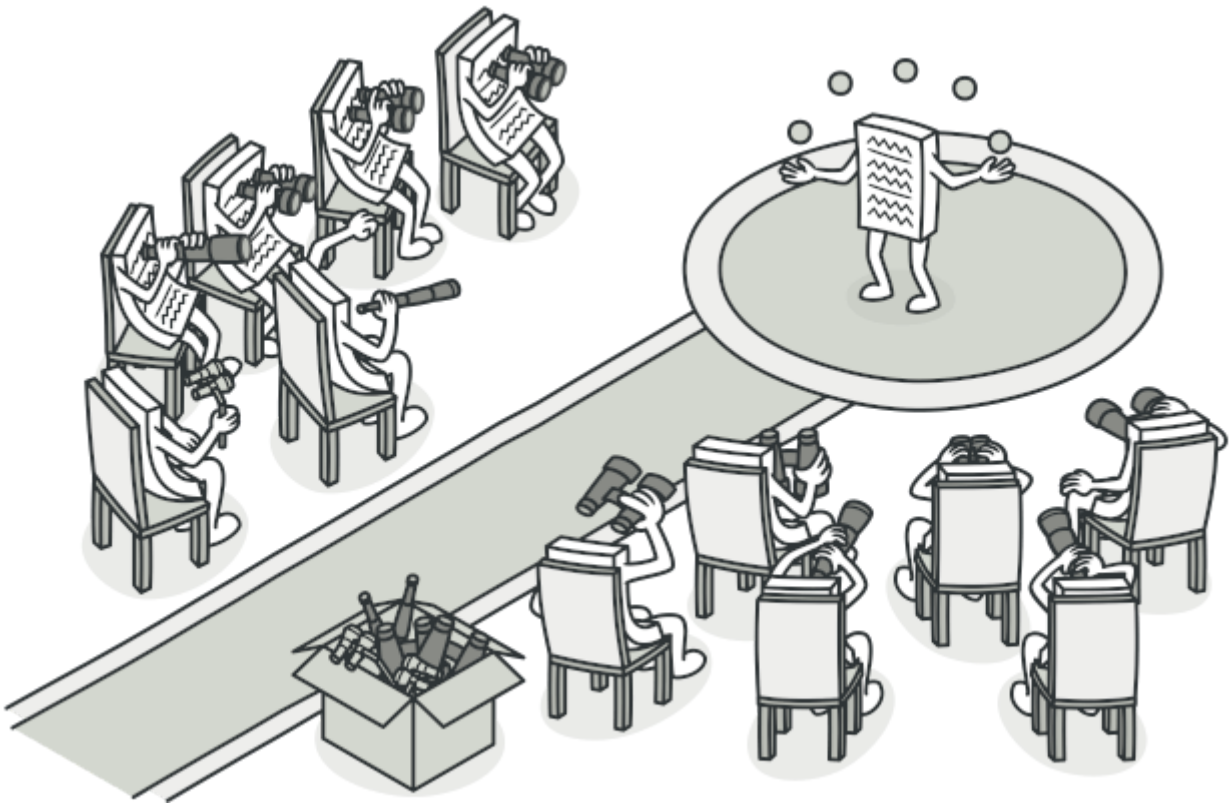

ComponentA
- m: Mediator
+ operationA()
ComponentB
- m: Mediator
+ operationB()
2
«interface»
Mediator
+ notify(sender)
1
1
ComponentC
- m: Mediator
+ operationC()
ComponentD
- m: Mediator
+ operationD()
3
ConcreteMediator
- componentA
- componentB
- componentC
- componentD
+ notify(sender)
+ reactOnA()
+ reactOnB()
+ reactOnC()
+ reactOnD()
m.notify(this)
4
if (sender == componentA)
reactOnA()

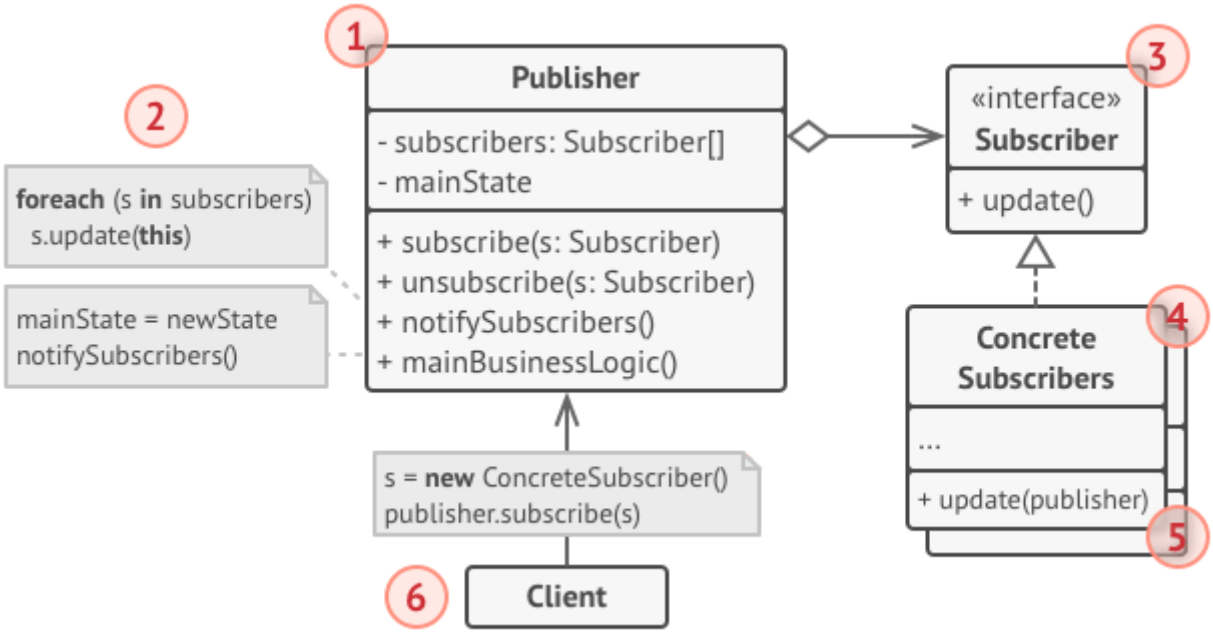
3.5 Memento

历史

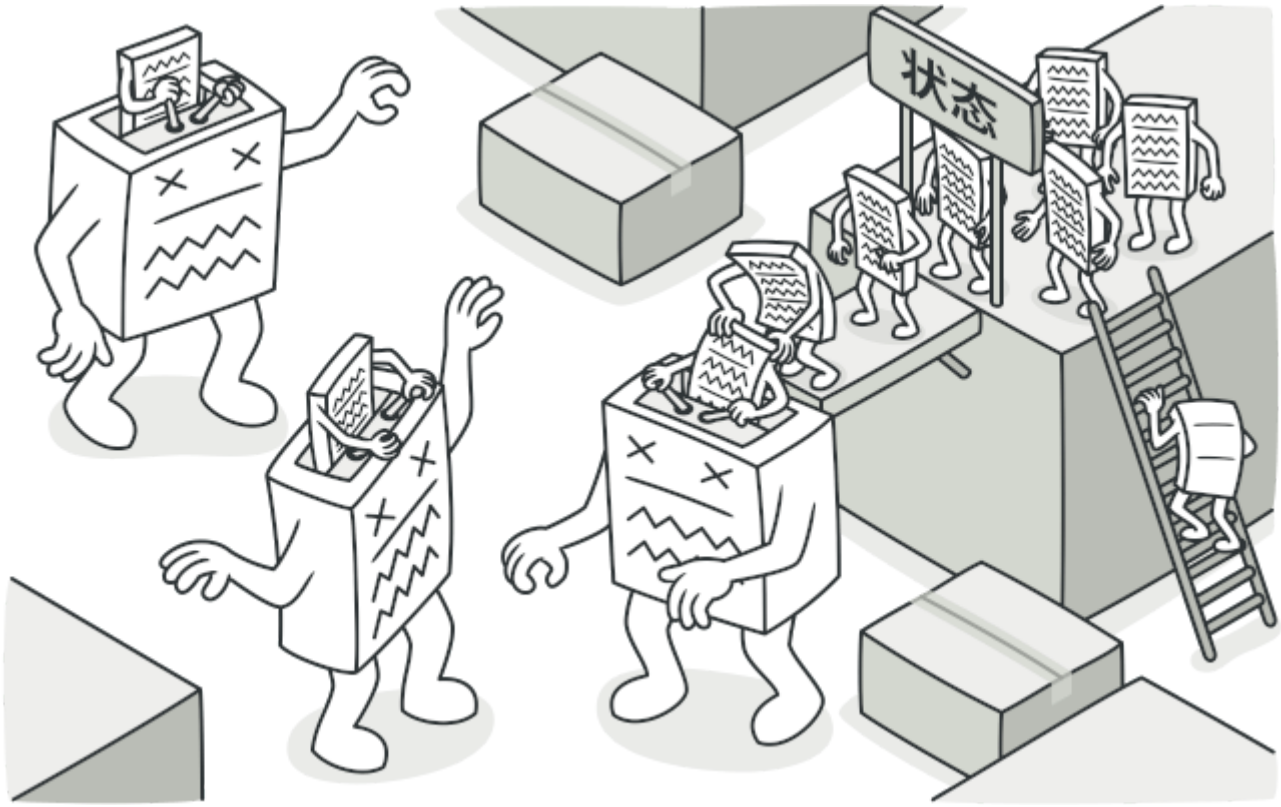

«interface»
Originator
save(): Memento
2
«interface»
Memento
+ restore()
Caretaker
- history: Memento[]
+ undo()
1
ConcreteOriginator
- state
+ save(): Memento
+ setState(state)
ConcreteMemento
- originator
- state
+ ConcreteMemento(originator, state)
+ restore()
3
originator.setState(state)

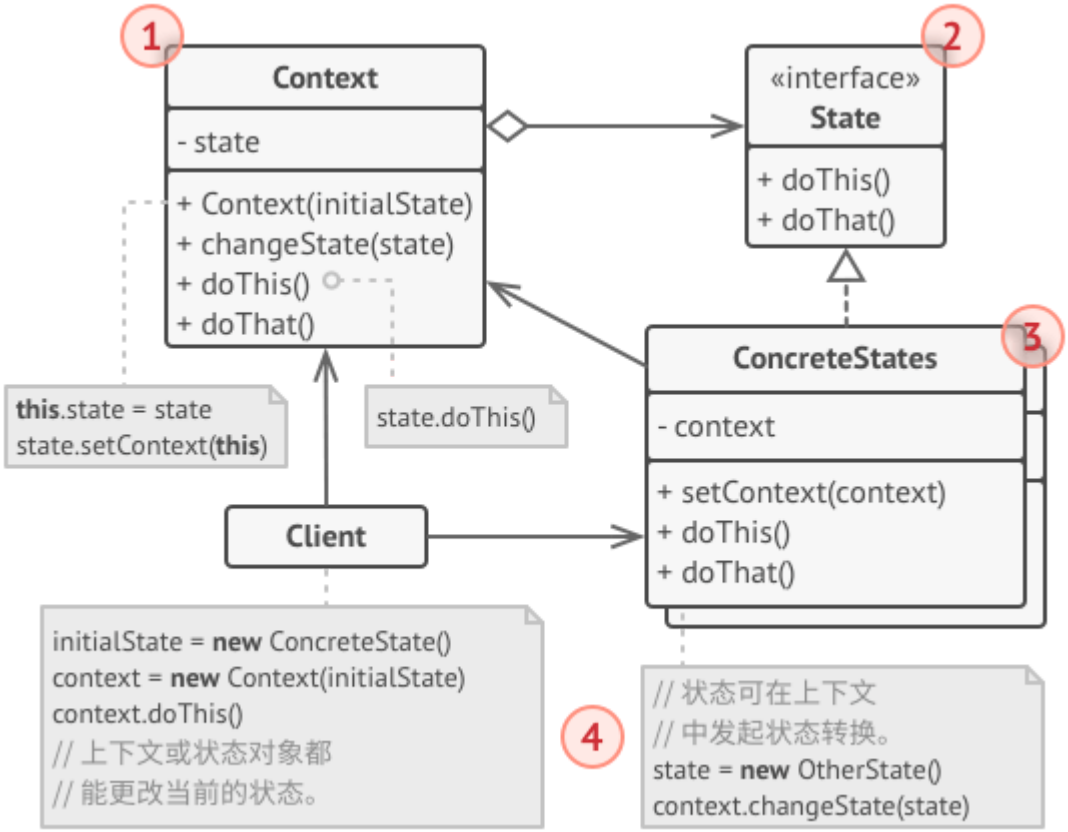
3.6 Observer



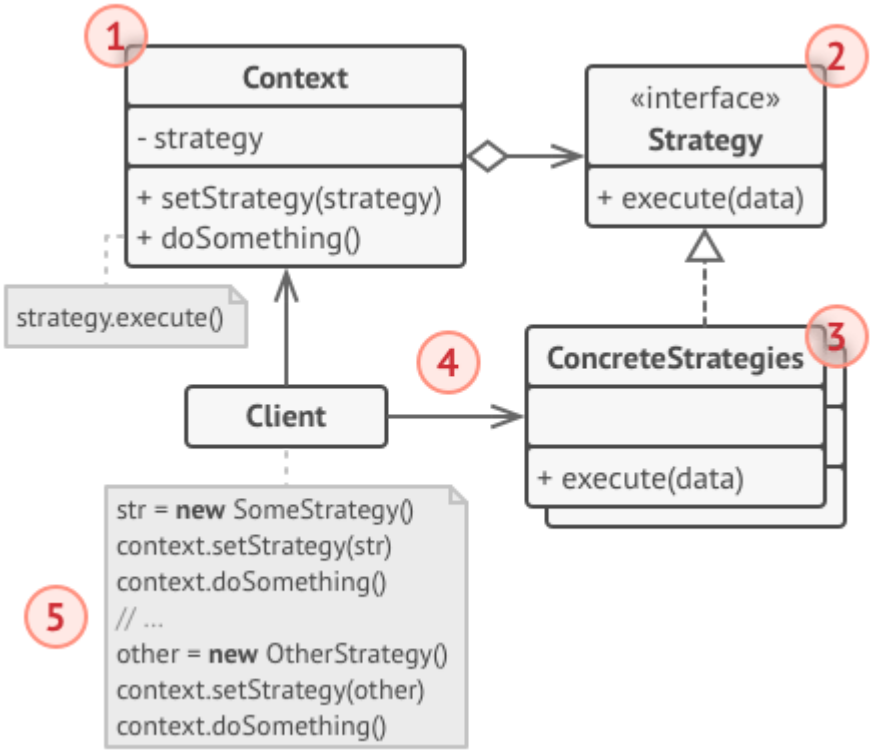

1
Publisher
- subscribers: Subscriber[]
- mainState
+ subscribe(s: Subscriber)
+ unsubscribe(s: Subscriber)
+ notifySubscribers()
+ mainBusinessLogic()
2
foreach (s in subscribers)
s.update(this)
mainState = newState
notifySubscribers()
3
«interface»
Subscriber
+ update()
4
Concrete
Subscribers
...
+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

3.7 State

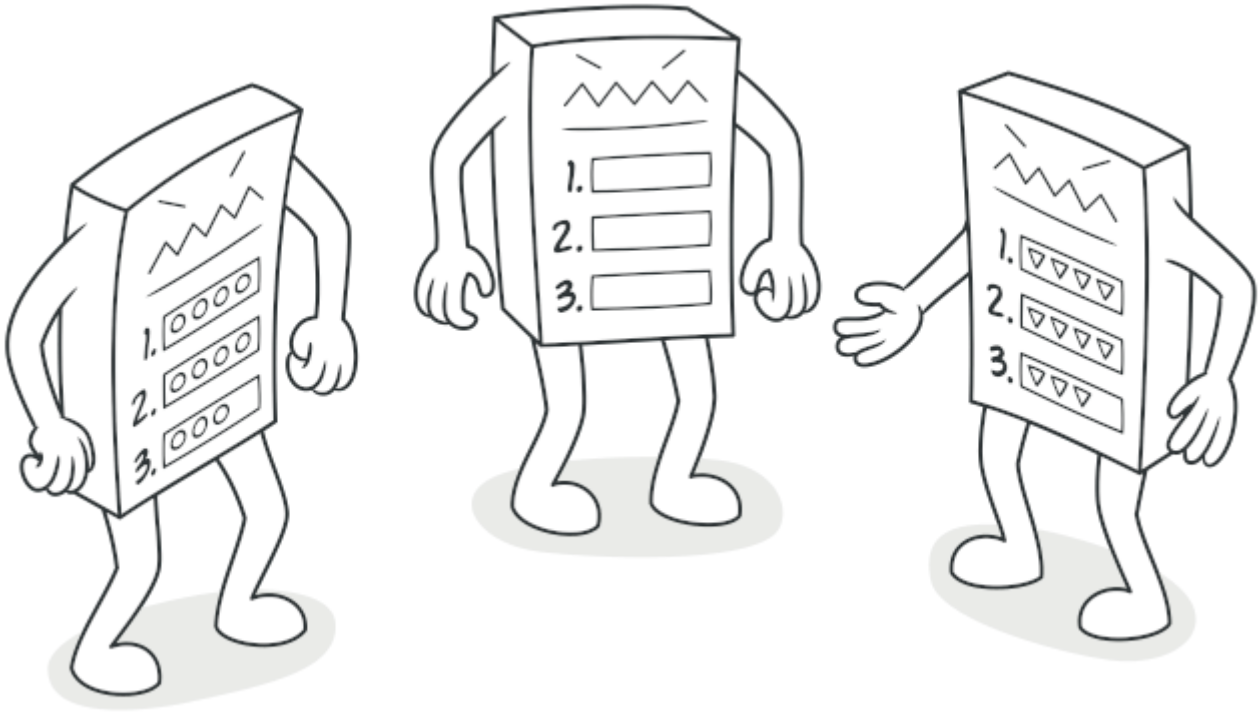

状态

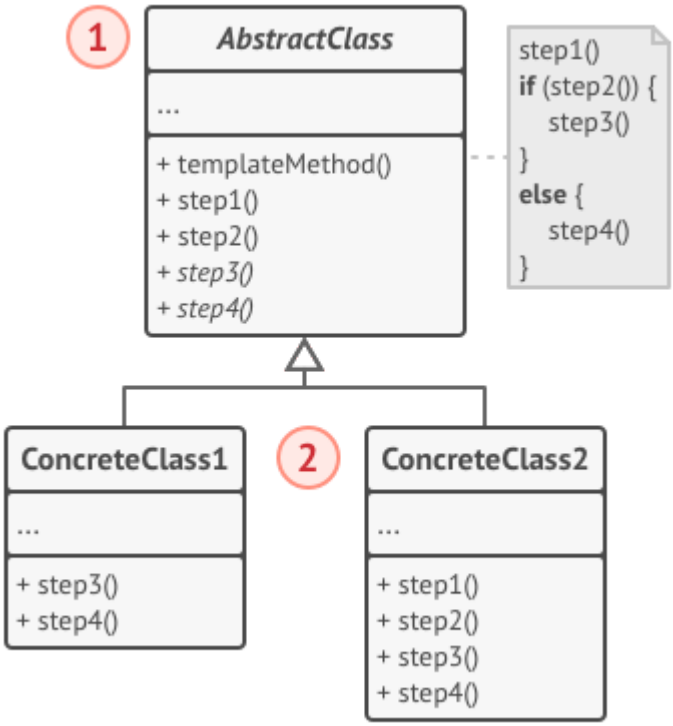

1
Context
- state
+ Context(initialState)
+ changeState(state)
+ doThis()
+ doThat()
2
«interface»
State
+ doThis()
+ doThat()
3
ConcreteStates
- context
+ setContext(context)
+ doThis()
+ doThat()
this.state = state
state.setContext(this)
state.doThis()
Client
initialState = new ConcreteState()
context = new Context(initialState)
context.doThis()
// 上下文或状态对象都
// 能更改当前的状态。
4
// 状态可在上下文
// 中发起状态转换。
state = new OtherState()
context.changeState(state)

3.8 Strategy

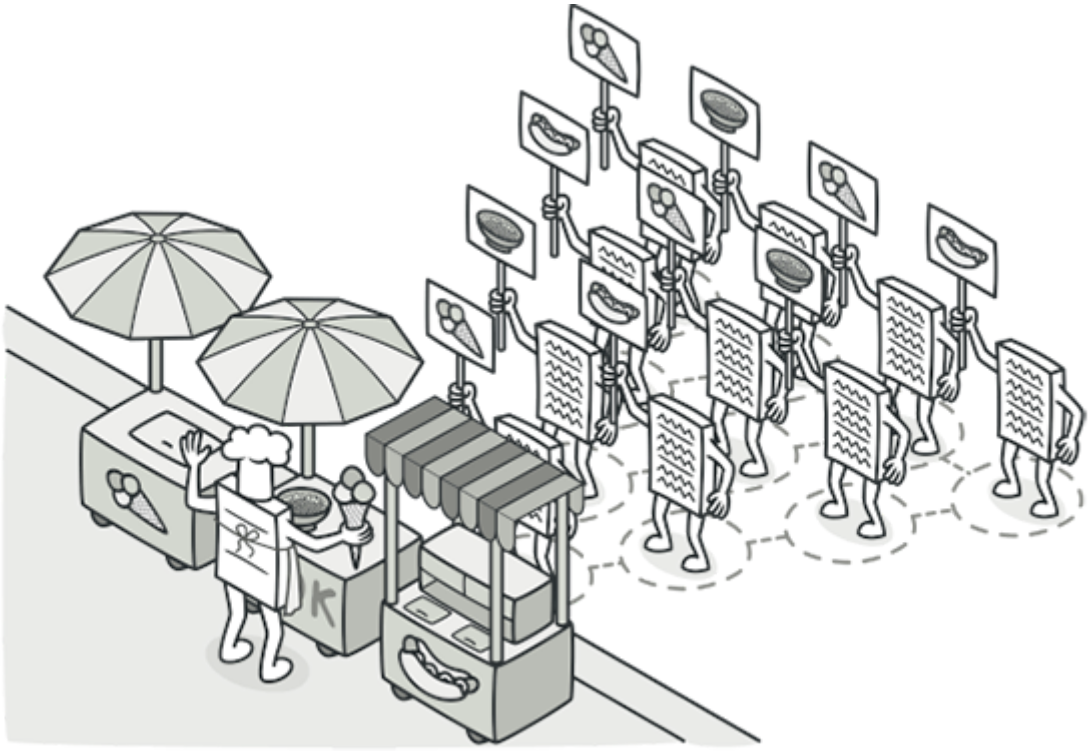

1
Context
- strategy
+ setStrategy(strategy)
+ doSomething()
2
«interface»
Strategy
+ execute(data)
strategy.execute()
4
Client
3
ConcreteStrategies
+ execute(data)
5
str = new SomeStrategy()
context.setStrategy(str)
context.doSomething()
// ...
other = new OtherStrategy()
context.setStrategy(other)
context.doSomething()

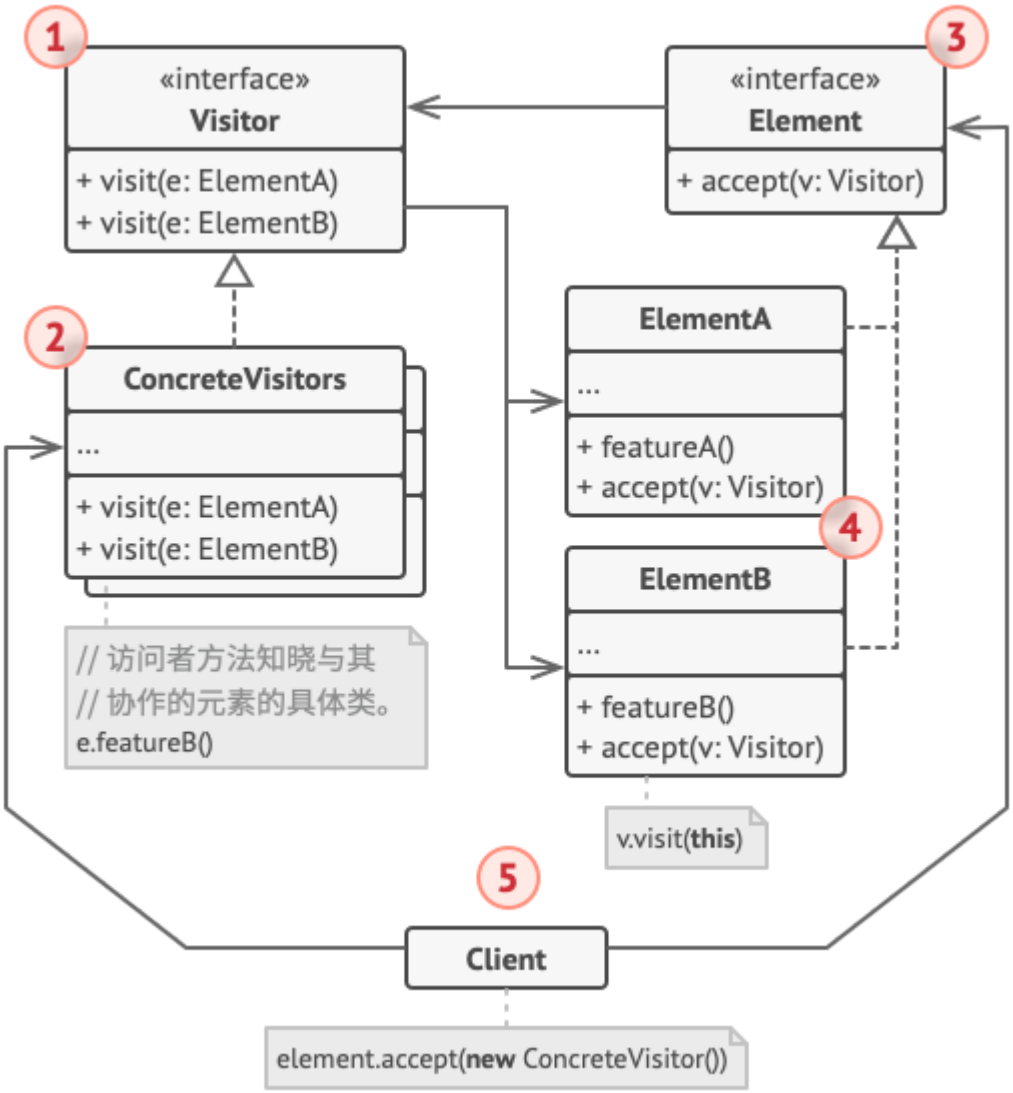
3.9 Template Method



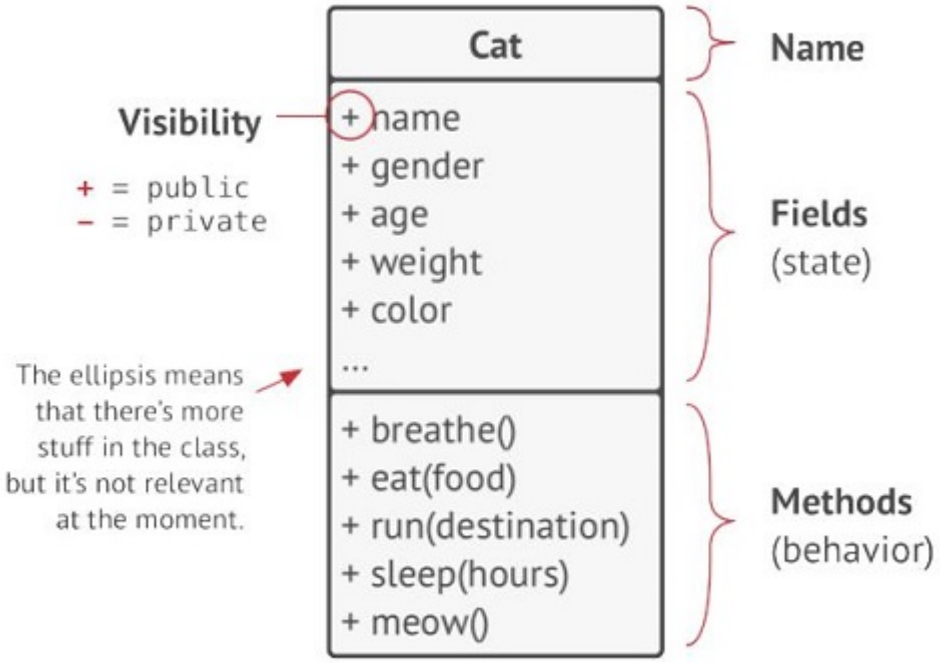


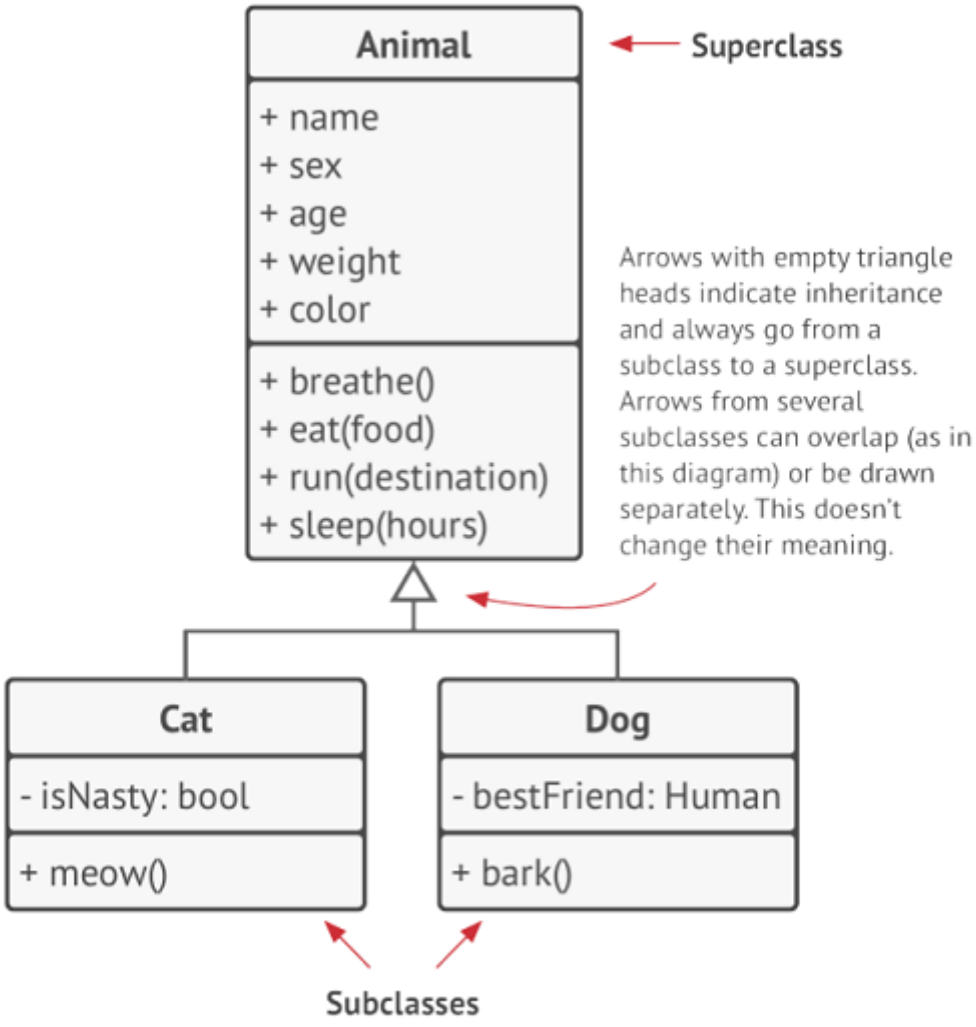
3.10 Vistor

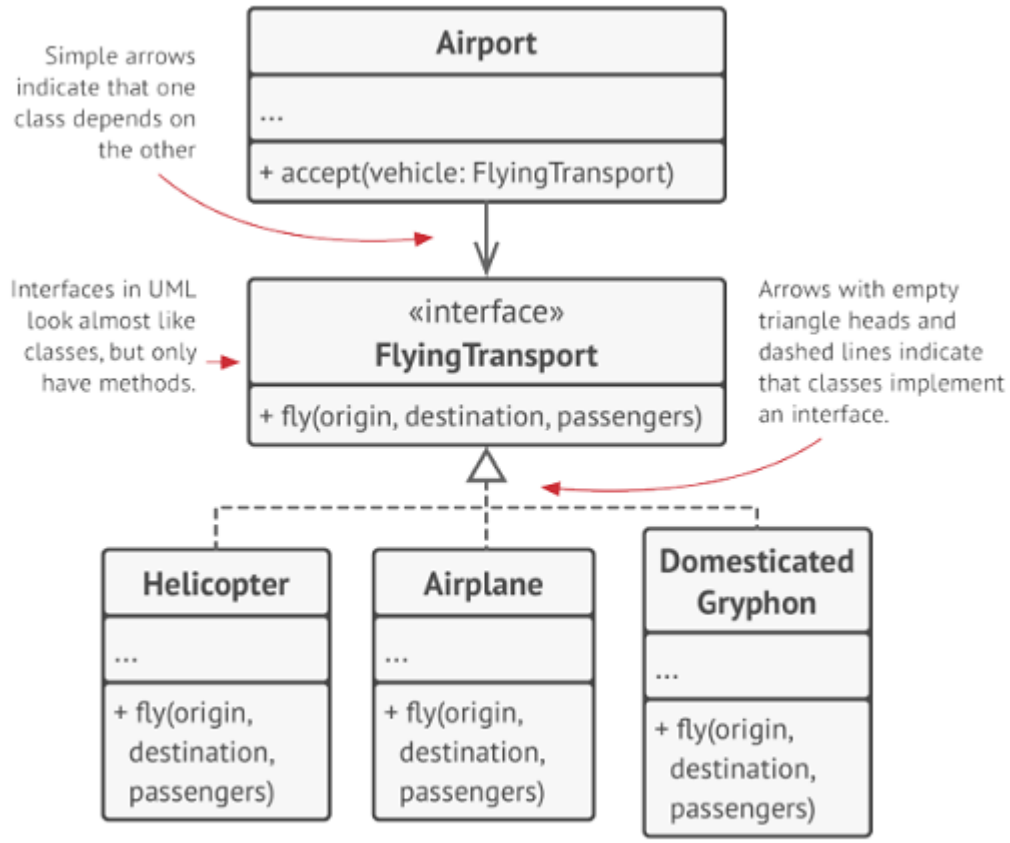



1
«interface»
Visitor
+ visit(e: ElementA)
+ visit(e: ElementB)
3
«interface»
Element
+ accept(v: Visitor)
2
ConcreteVisitors
...
+ visit(e: ElementA)
+ visit(e: ElementB)
// 访问者方法知晓与其
// 协作的元素的具体类。
e.featureB()
ElementA
...
+ featureA()
+ accept(v: Visitor)
4
ElementB
...
+ featureB()
+ accept(v: Visitor)
v.visit(this)
5
Client
element.accept(new ConcreteVisitor())

4 UML


Cat
Name
Visibility
+ = public
– = private
+ name
+ gender
+ age
+ weight
+ color
...
Fields
(state)
The ellipsis means that there's more stuff in the class, but it's not relevant at the moment.
+ breathe()
+ eat(food)
+ run(destination)
+ sleep(hours)
+ meow()
Methods
(behavior)


Animal
Superclass
+ name
+ sex
+ age
+ weight
+ color
+ breathe()
+ eat(food)
+ run(destination)
+ sleep(hours)
Arrows with empty triangle heads indicate inheritance and always go from a subclass to a superclass. Arrows from several subclasses can overlap (as in this diagram) or be drawn separately. This doesn't change their meaning.
Cat
- isNasty: bool
+ meow()
Dog
- bestFriend: Human
+ bark()
Subclasses



UML diagram of several classes implementing an interface.

UML Association. Professor communicates with students.

UML Dependency. Professor depends on salary.

UML Composition. University consists of departments.

UML Aggregation. Department contains professors.