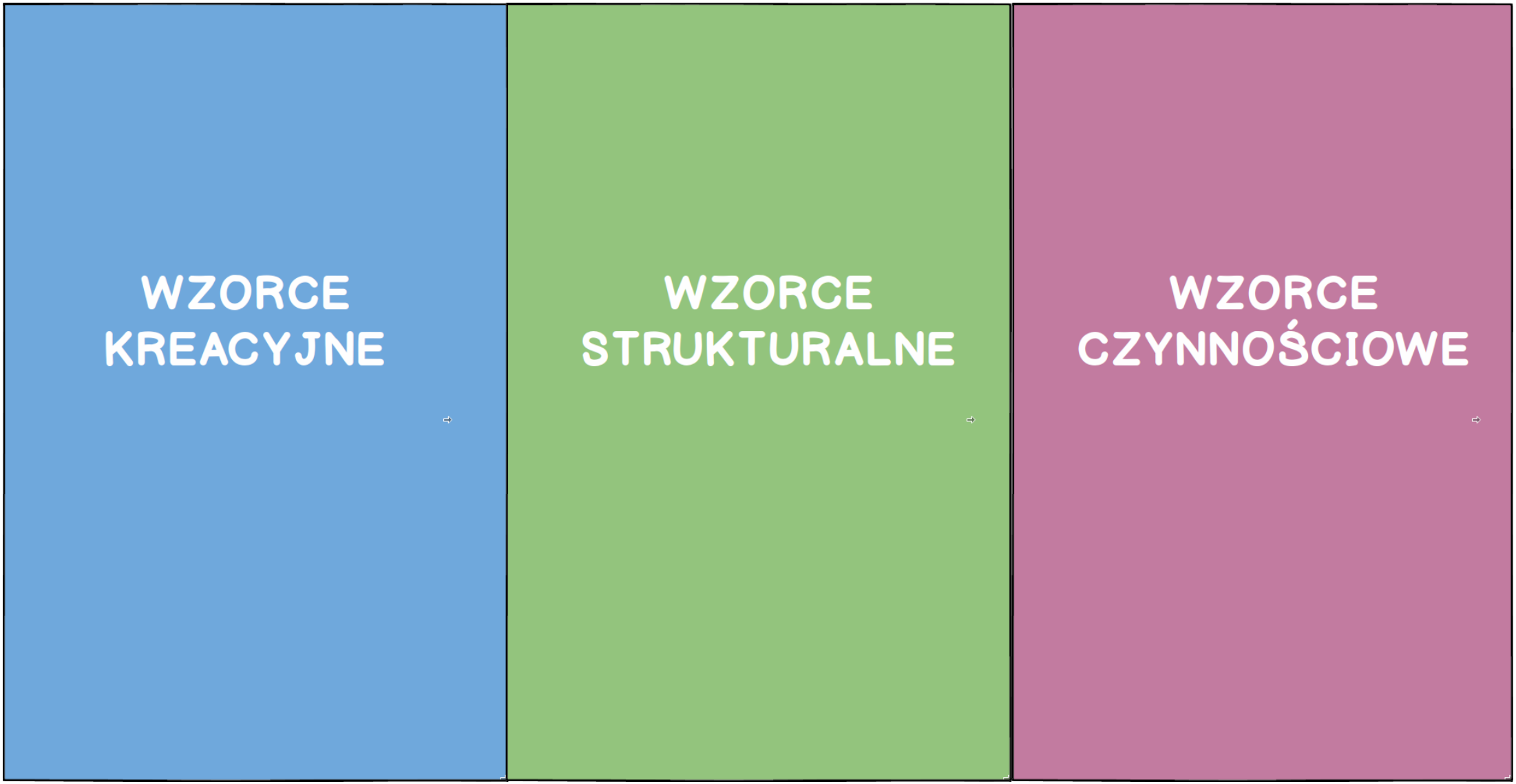
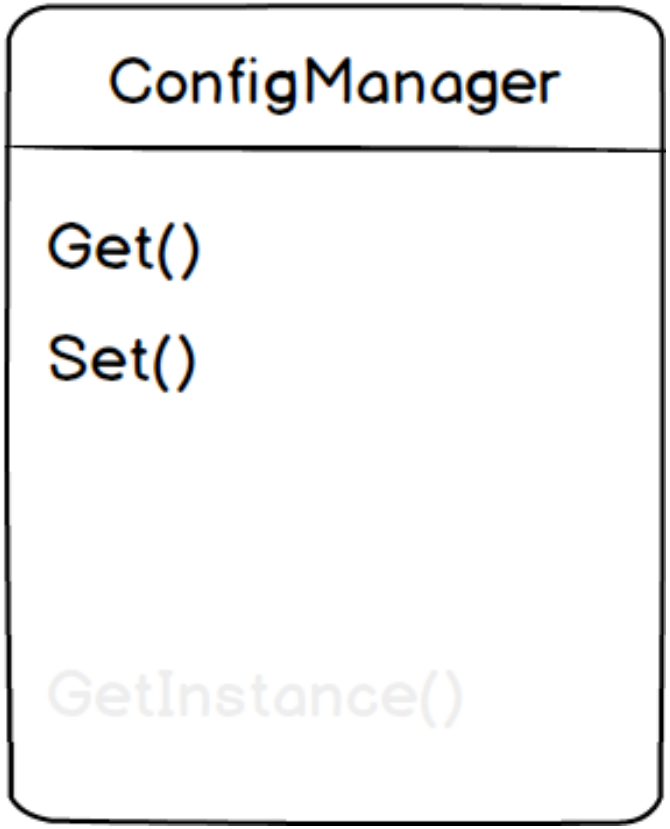
WZORCE PROJEKTOWE

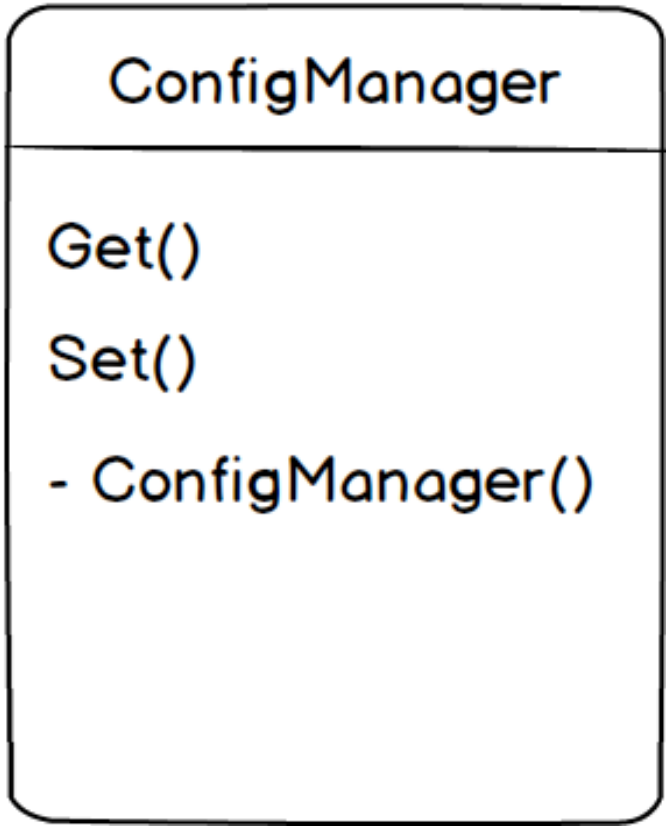


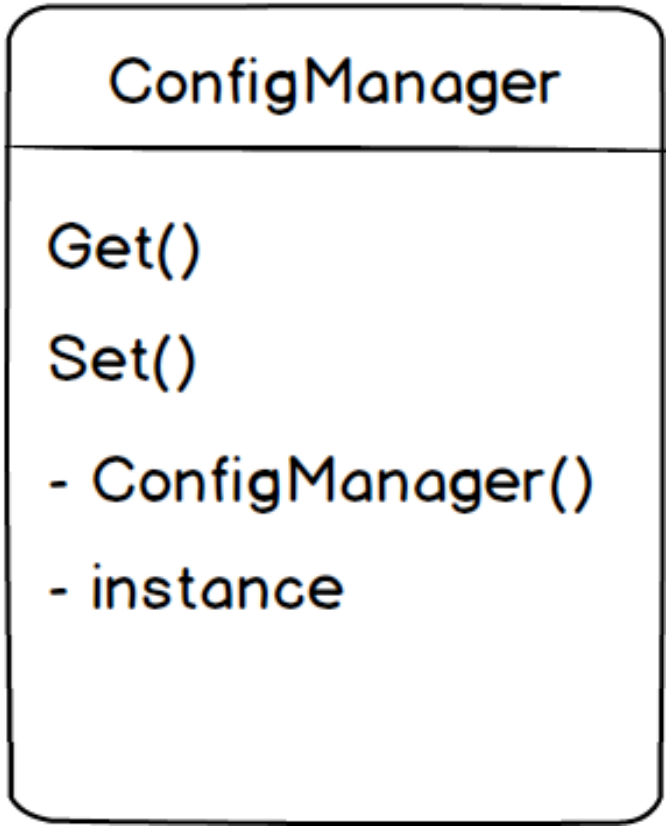
WZORCE KREACYJNE

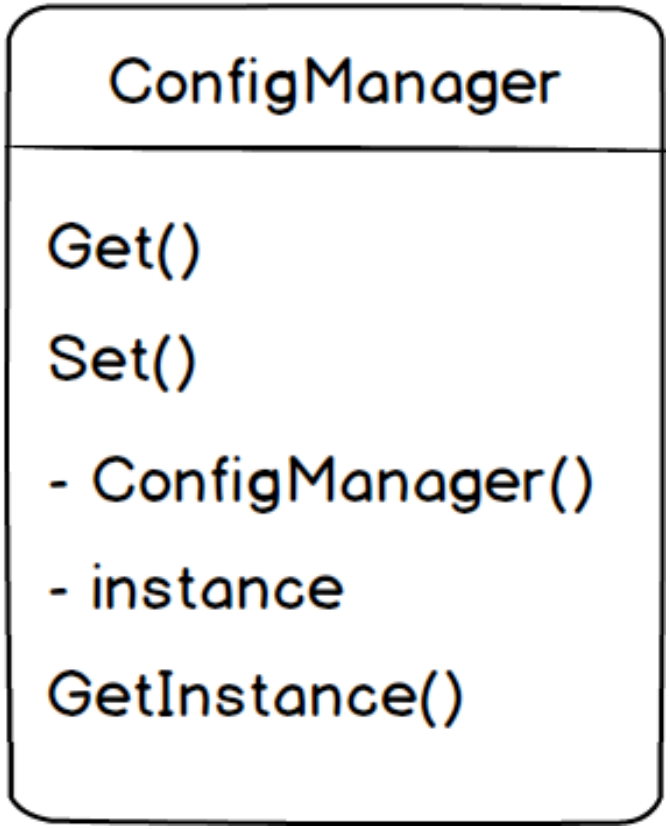
Singleton Pattern

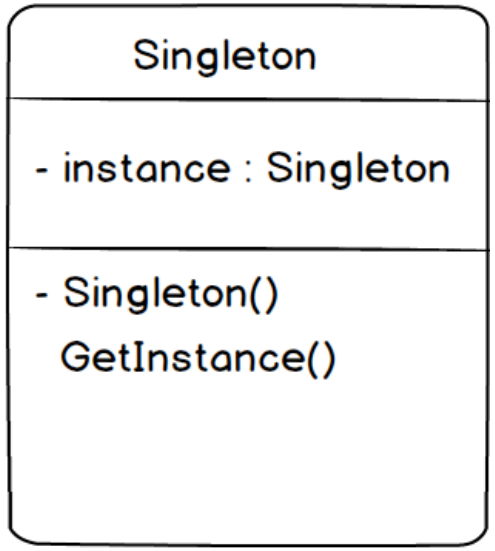
Zapewnia, że klasa posiada
pojedynczą instancję


ConfigManager
Get()
Set()
GetInstance()


ConfigManager
Get()
Set()
- ConfigManager()


ConfigManager
Get()
Set()
- ConfigManager()
- instance

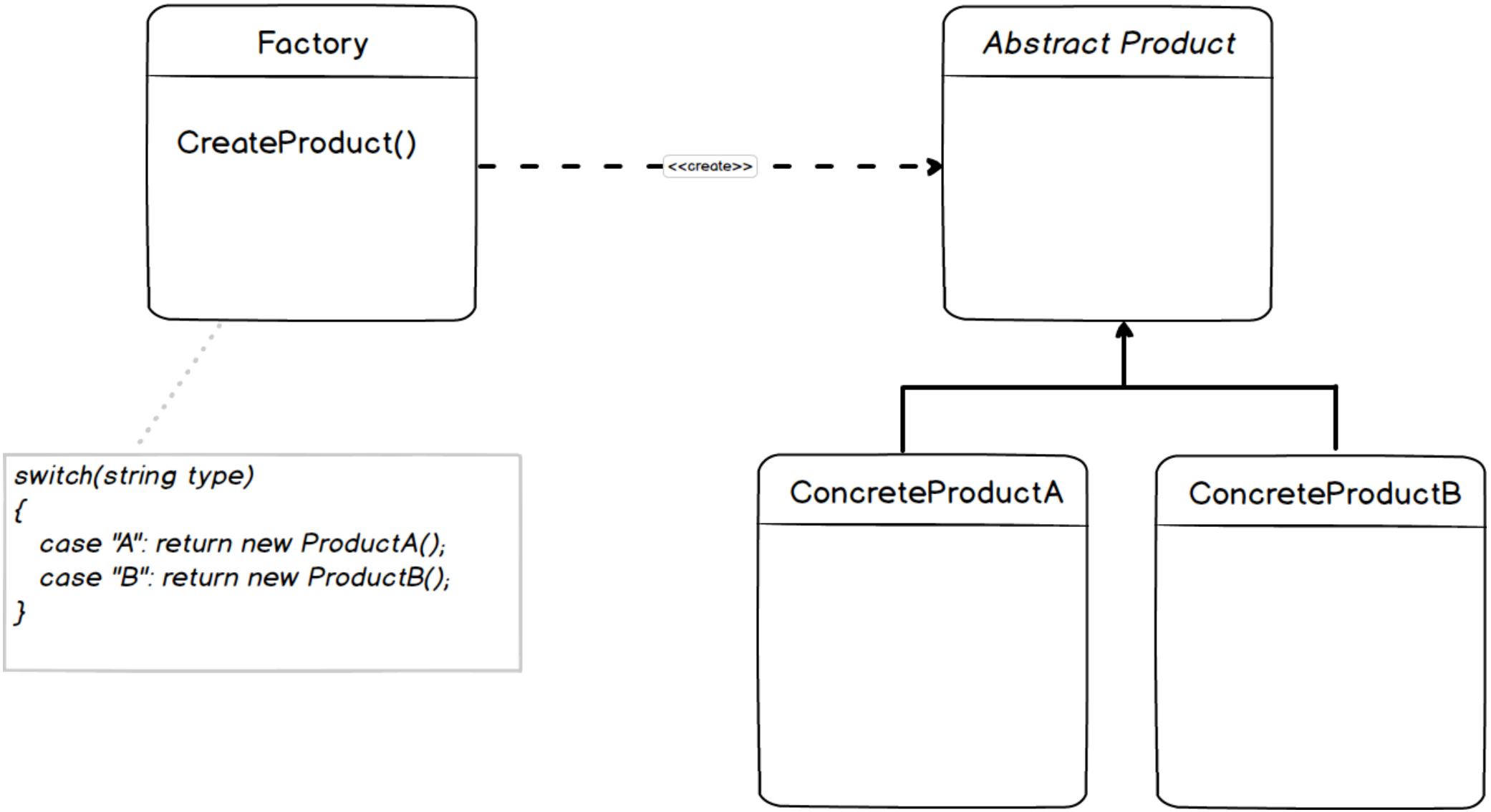

ConfigManager
Get()
Set()
- ConfigManager()
- instance
GetInstance()



Pozwala zachować pewność. że istnieje wyłącznie **jedna instancja** danej klasy.

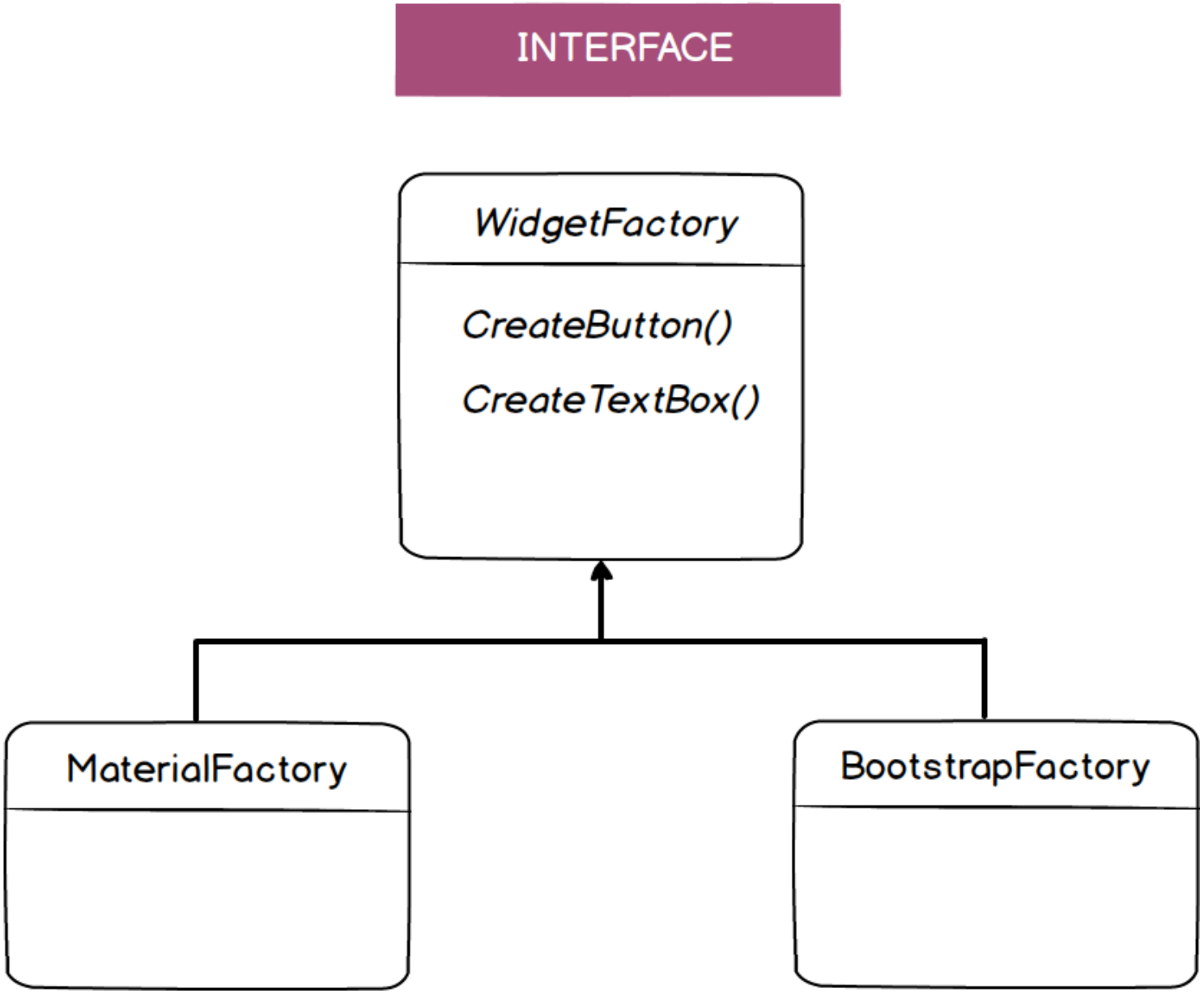
Simple Factory Pattern

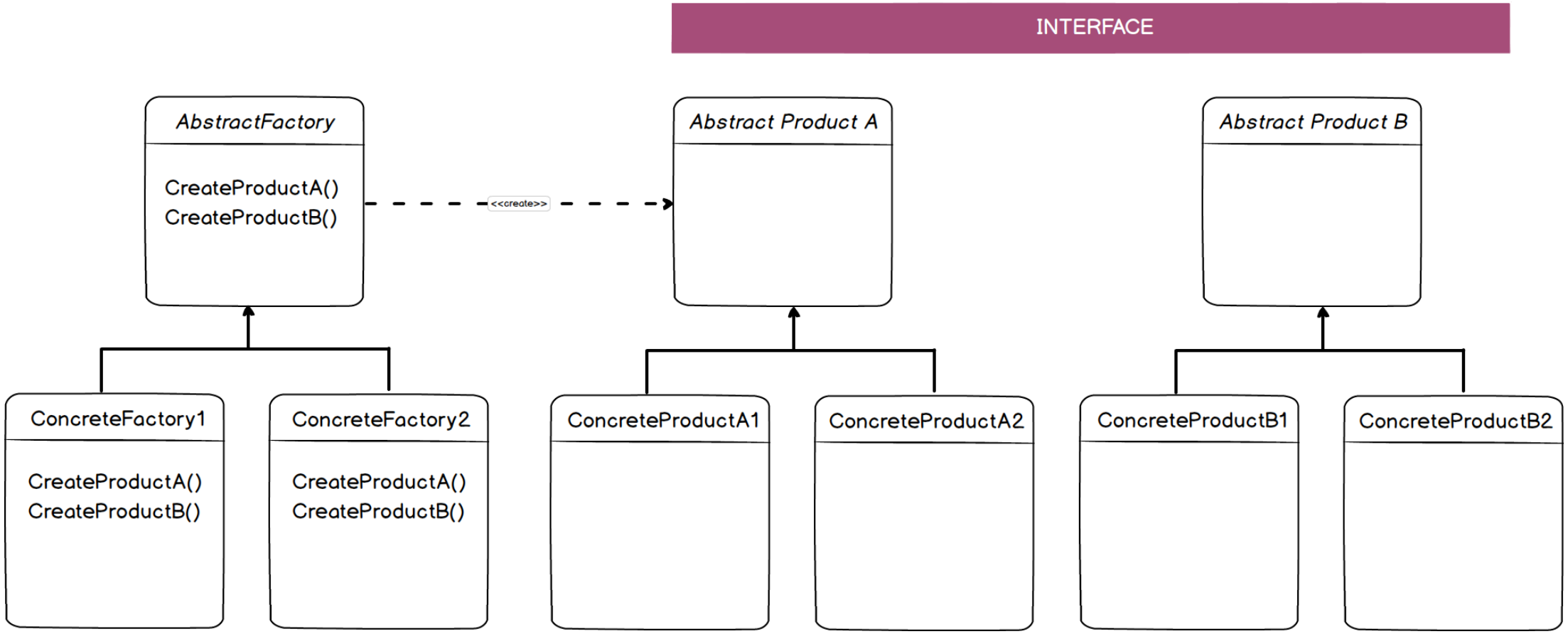
Zapewnia tworzenie obiektu na podstawie wielu **warunków**


Factory
CreateProduct()
<<create>>
Abstract Product
switch(string type)
{
case "A": return new ProductA();
case "B": return new ProductB();
}
ConcreteProductA
ConcreteProductB

Abstract Factory Pattern

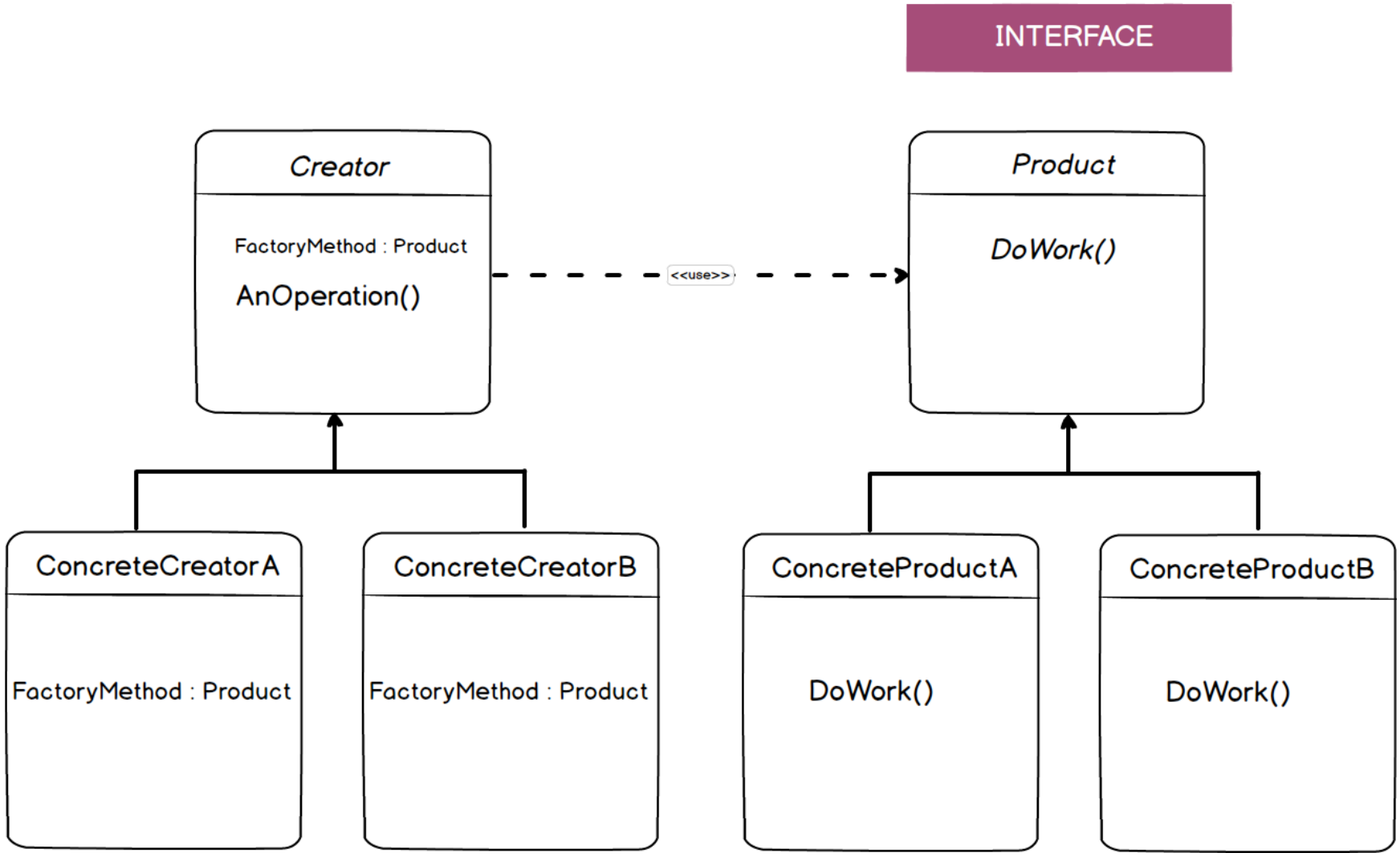
Zapewnia interfejs do tworzenia
rodziny powiązanych obiektów


INTERFACE
WidgetFactory
CreateButton()
CreateTextBox()
MaterialFactory
BootstrapFactory


INTERFACE
AbstractFactory
CreateProductA()
CreateProductB()
<<create>>
Abstract Product A
Abstract Product B
ConcreteFactory1
CreateProductA()
CreateProductB()
ConcreteFactory2
CreateProductA()
CreateProductB()
ConcreteProductA1
ConcreteProductA2
ConcreteProductB1
ConcreteProductB2

Factory Method Pattern

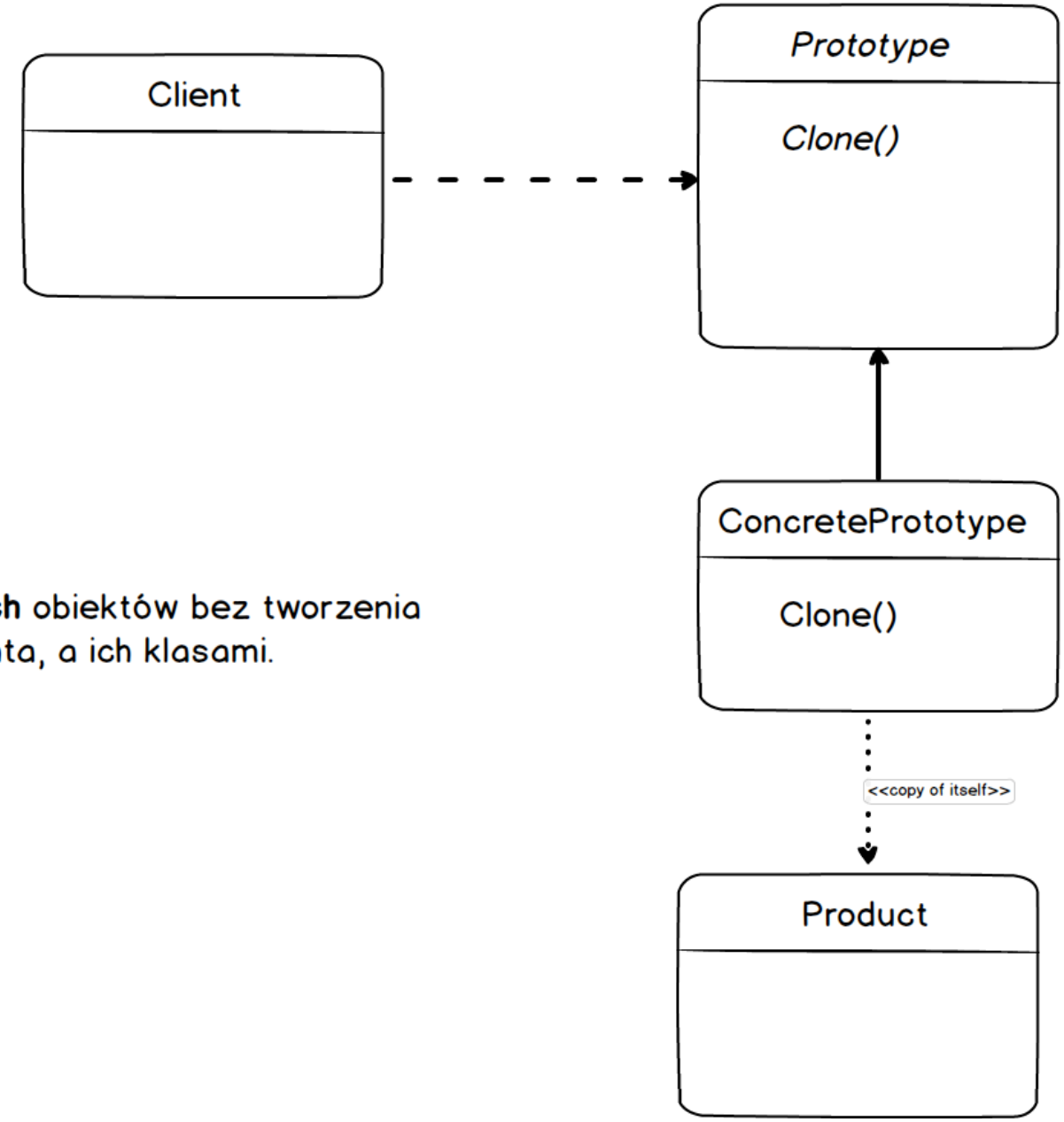
Odkłada tworzenie obiektu
do **podklas**


INTERFACE
Creator
FactoryMethod : Product
AnOperation()
<<use>>
Product
DoWork()
ConcreteCreatorA
FactoryMethod : Product
ConcreteCreatorB
FactoryMethod : Product
ConcreteProductA
DoWork()
ConcreteProductB
DoWork()

Prototype Pattern

Tworzy nowe obiekty
kopiując **istniejący** obiekt

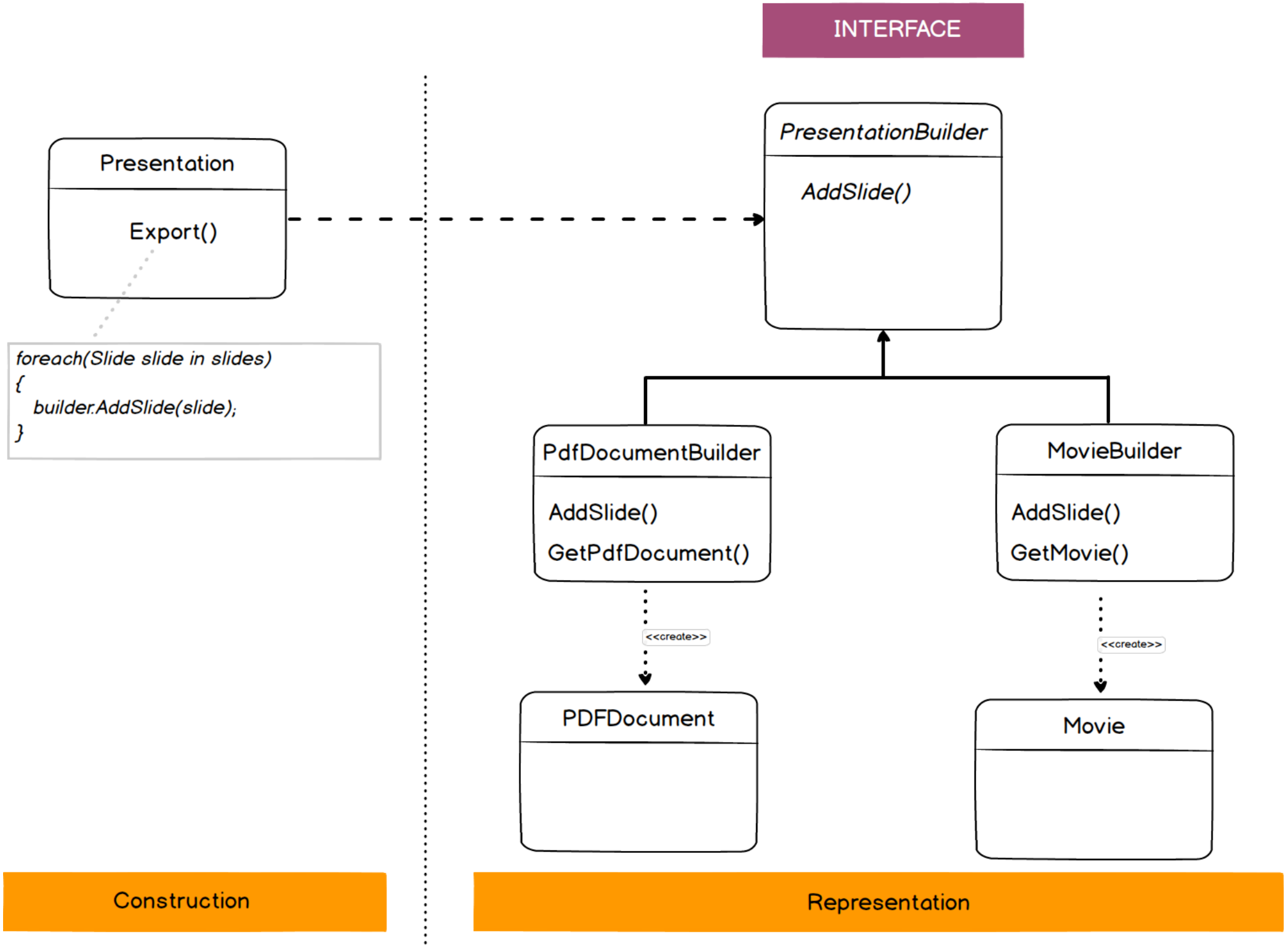
INTERFACE

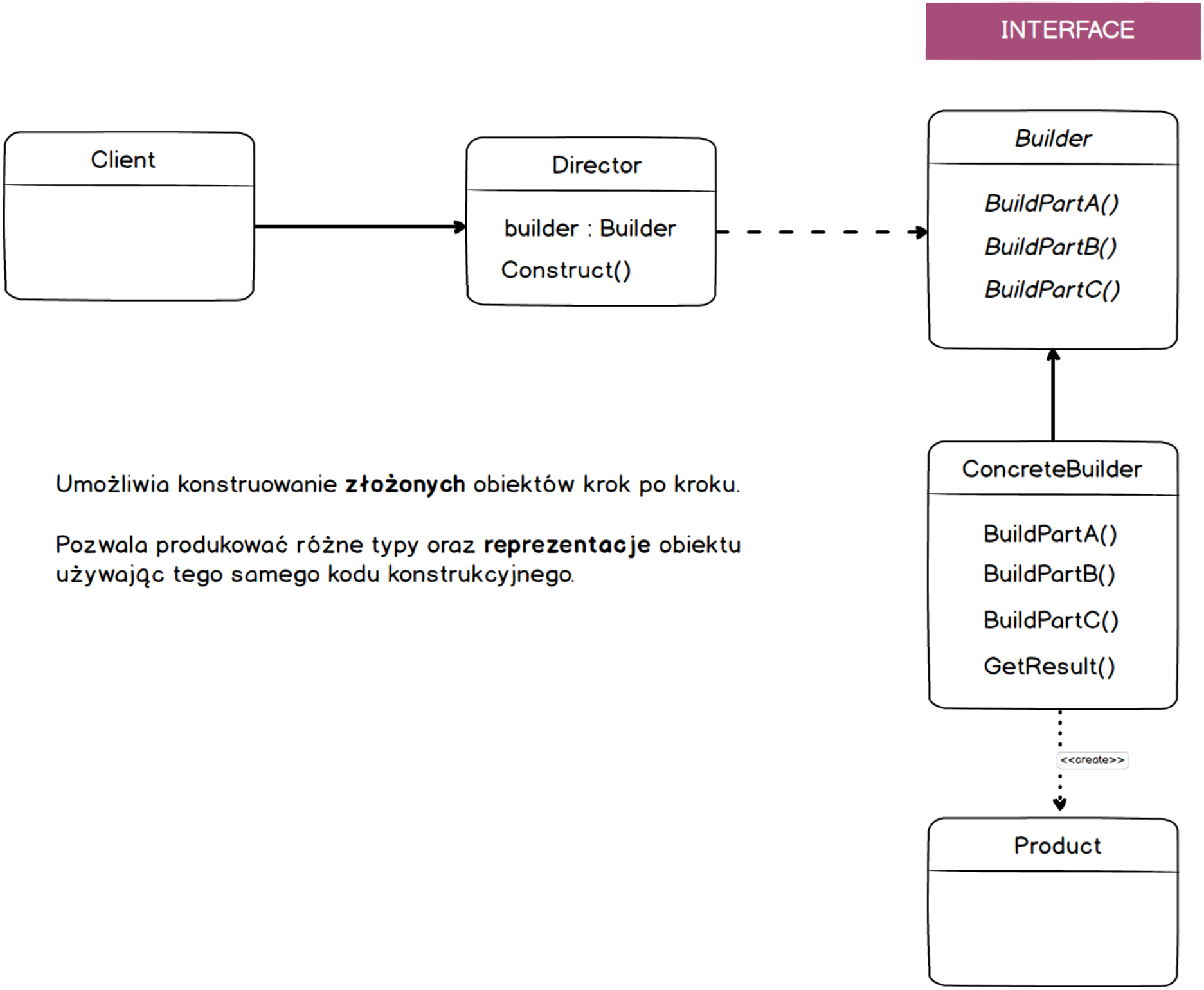


Umożliwia kopiowanie **istniejących** obiektów bez tworzenia zależności pomiędzy kodem klienta, a ich klasami.

Builder Pattern

Oddziela **konstrukcję** obiektu
od jego **reprezentacji**


INTERFACE
Presentation
Export()
foreach(Slide slide in slides)
{
builder.AddSlide(slide);
}
PresentationBuilder
AddSlide()
PdfDocumentBuilder
AddSlide()
GetPdfDocument()
MovieBuilder
AddSlide()
GetMovie()
<<create>>
<<create>>
PDFDocument
Movie
Construction
Representation



Umożliwia konstruowanie **złożonych** obiektów krok po kroku.

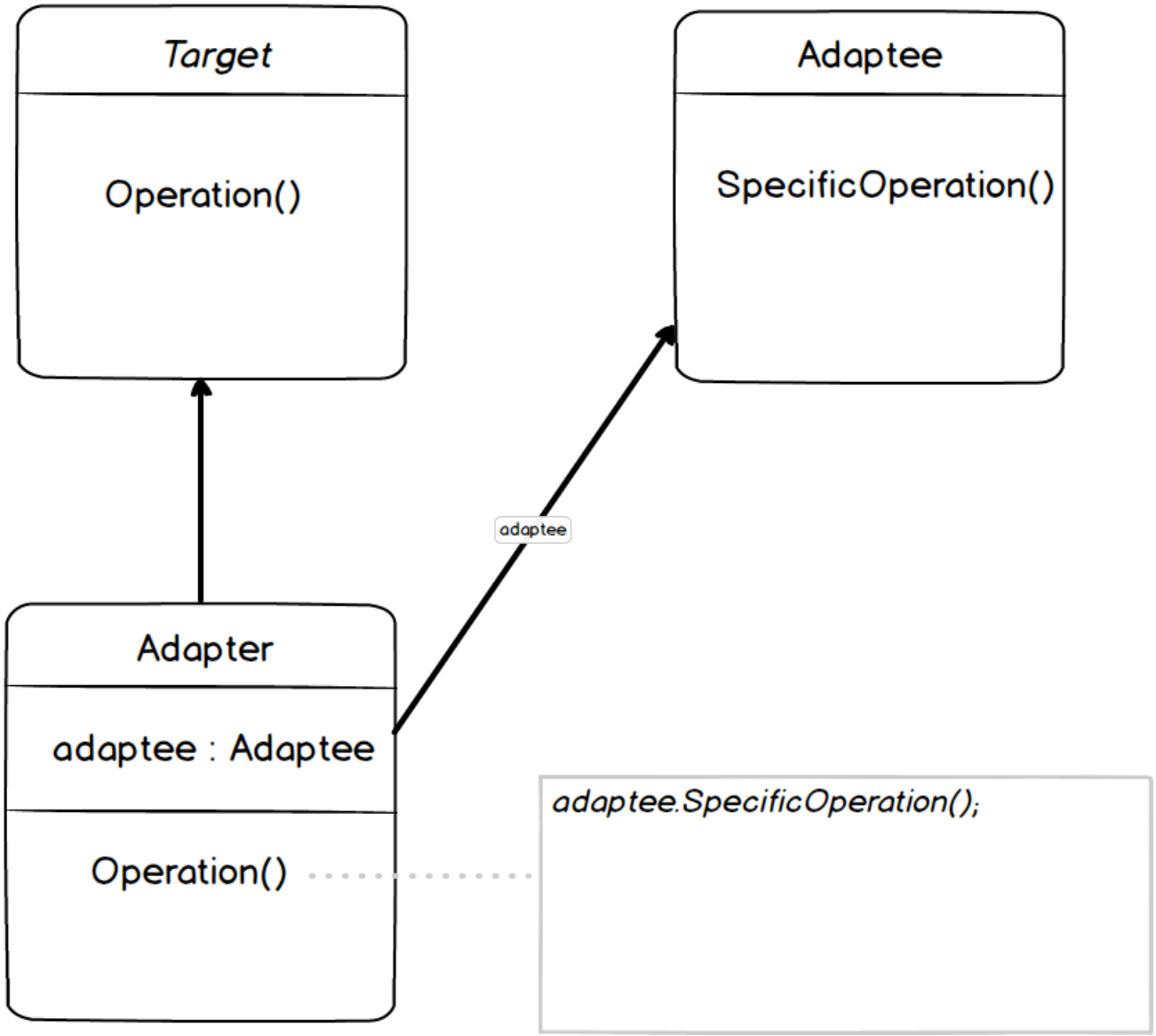
Pozwala produkować różne typy oraz **reprezentacje** obiektu używając tego samego kodu konstrukcyjnego.

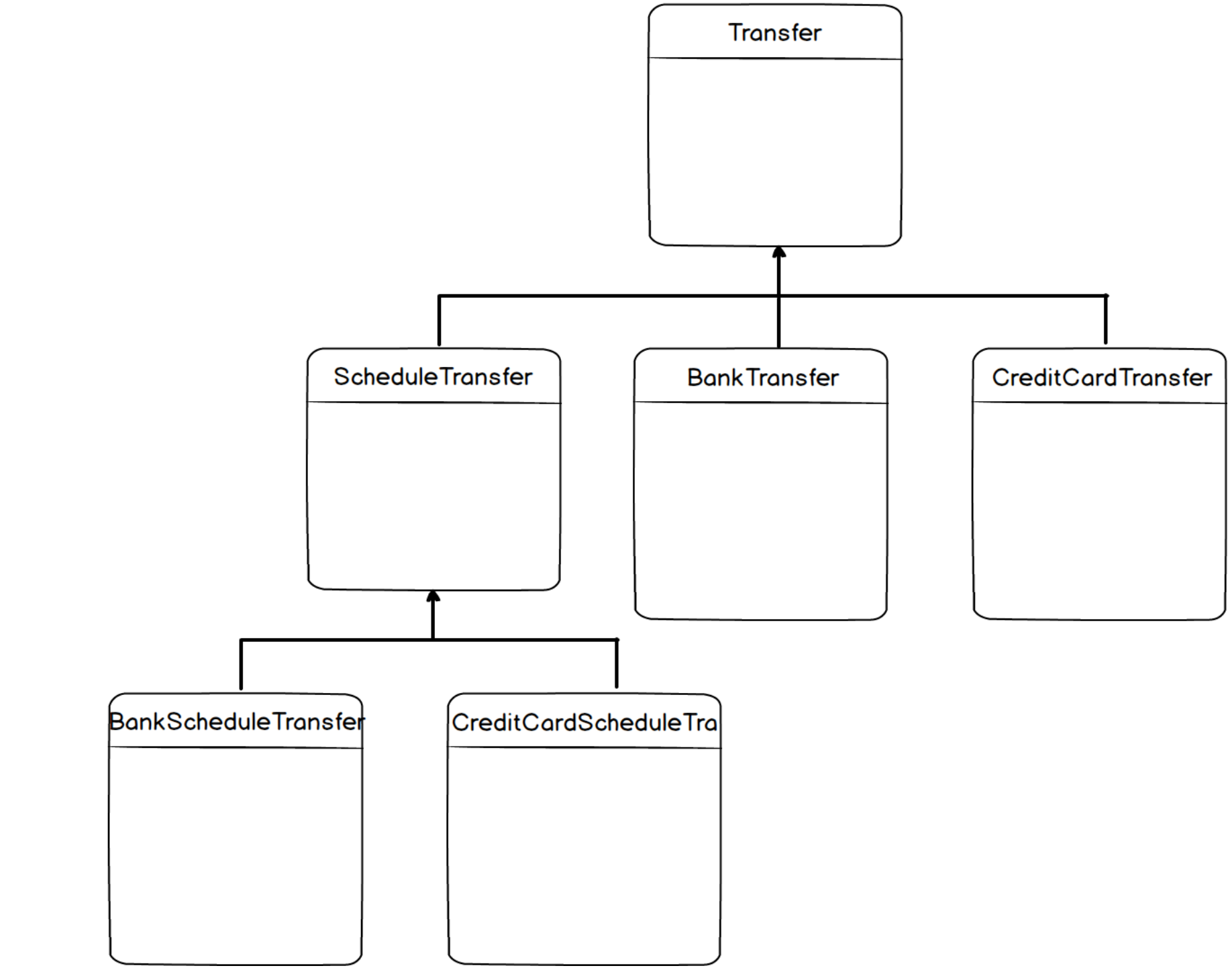
WZORCE STRUKTURALNE

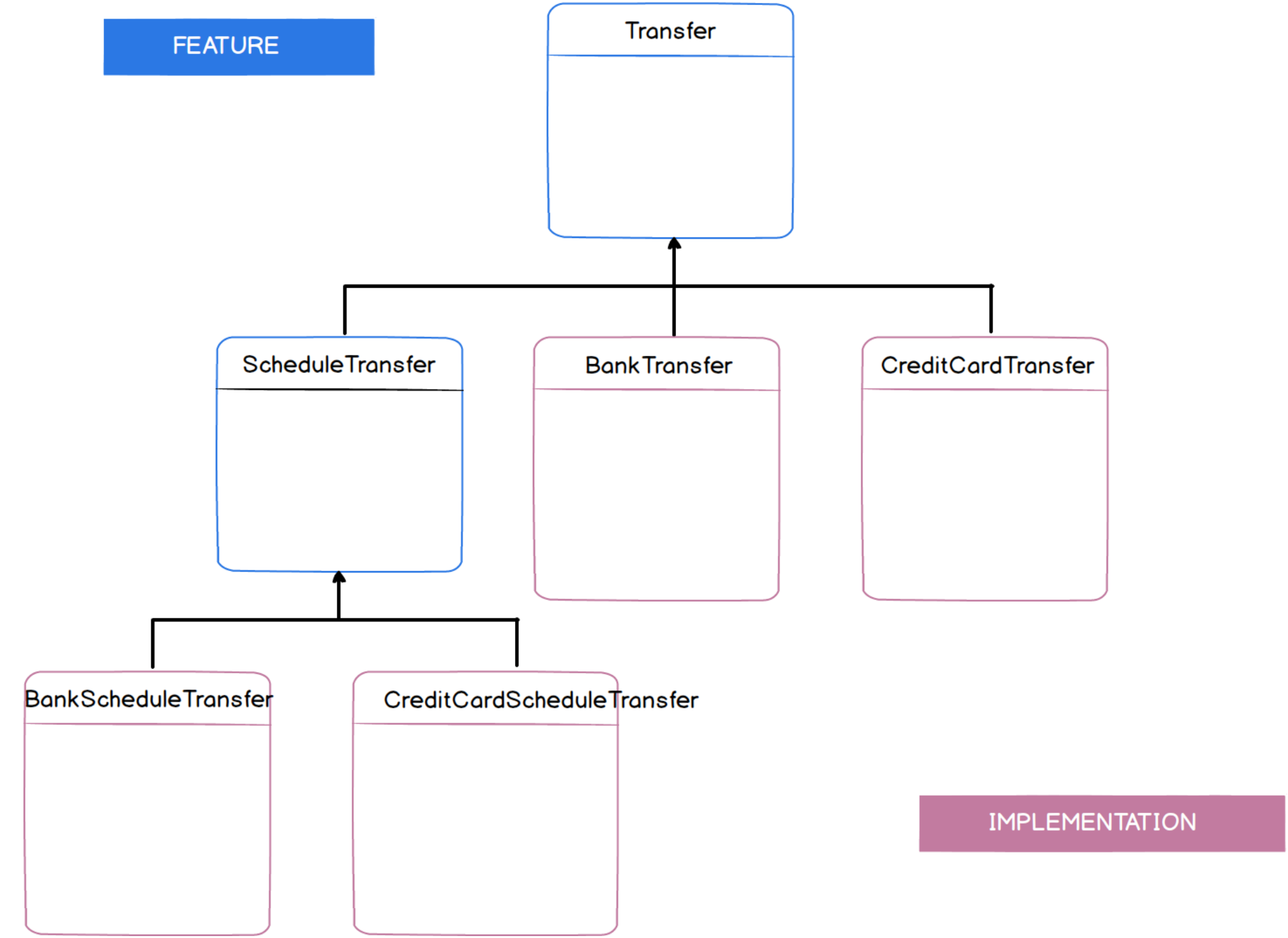
Adapter Pattern

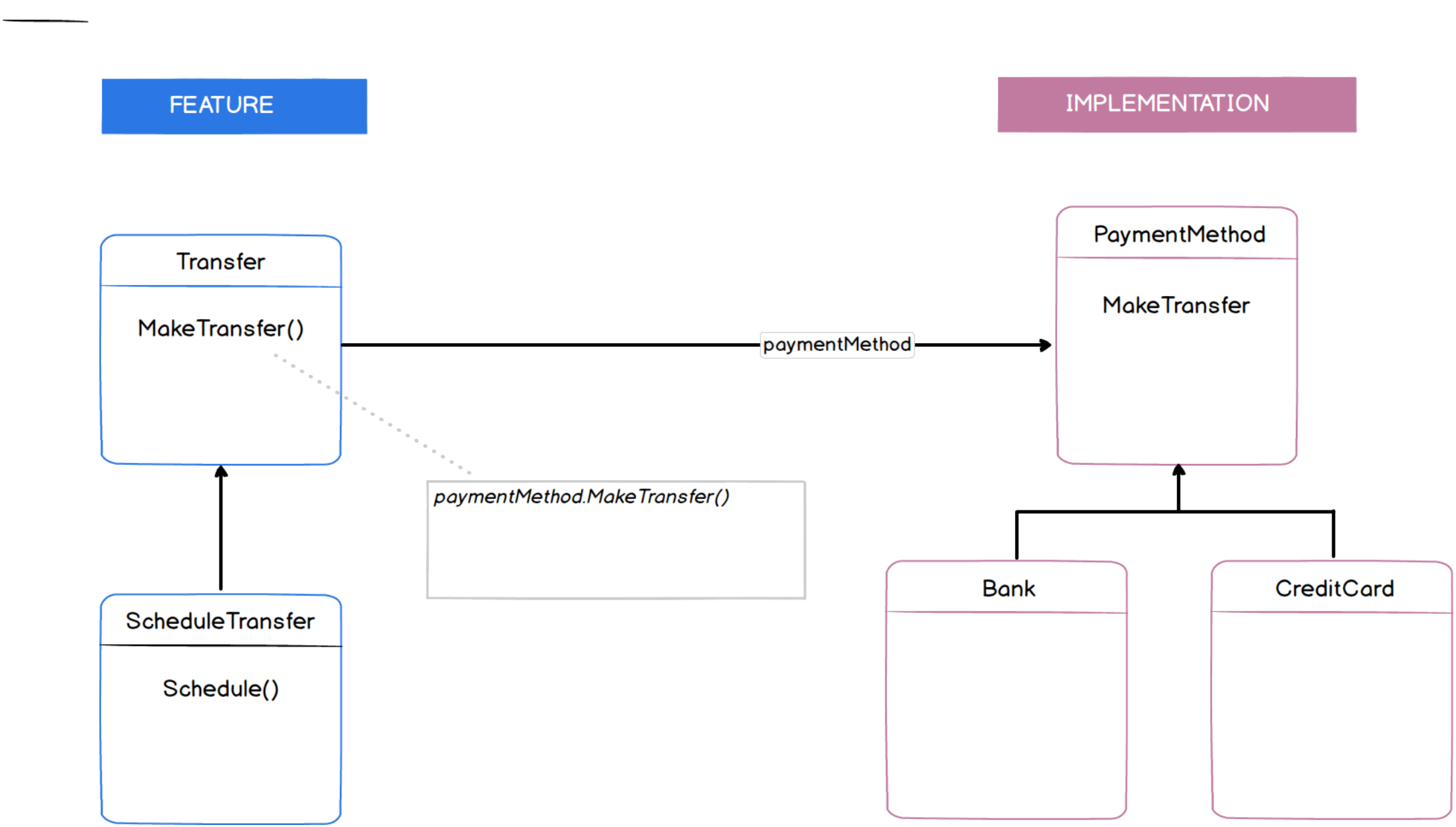
Pozwala na współdziałanie ze sobą
obiektów o **niekompatybilnych** interfejsach

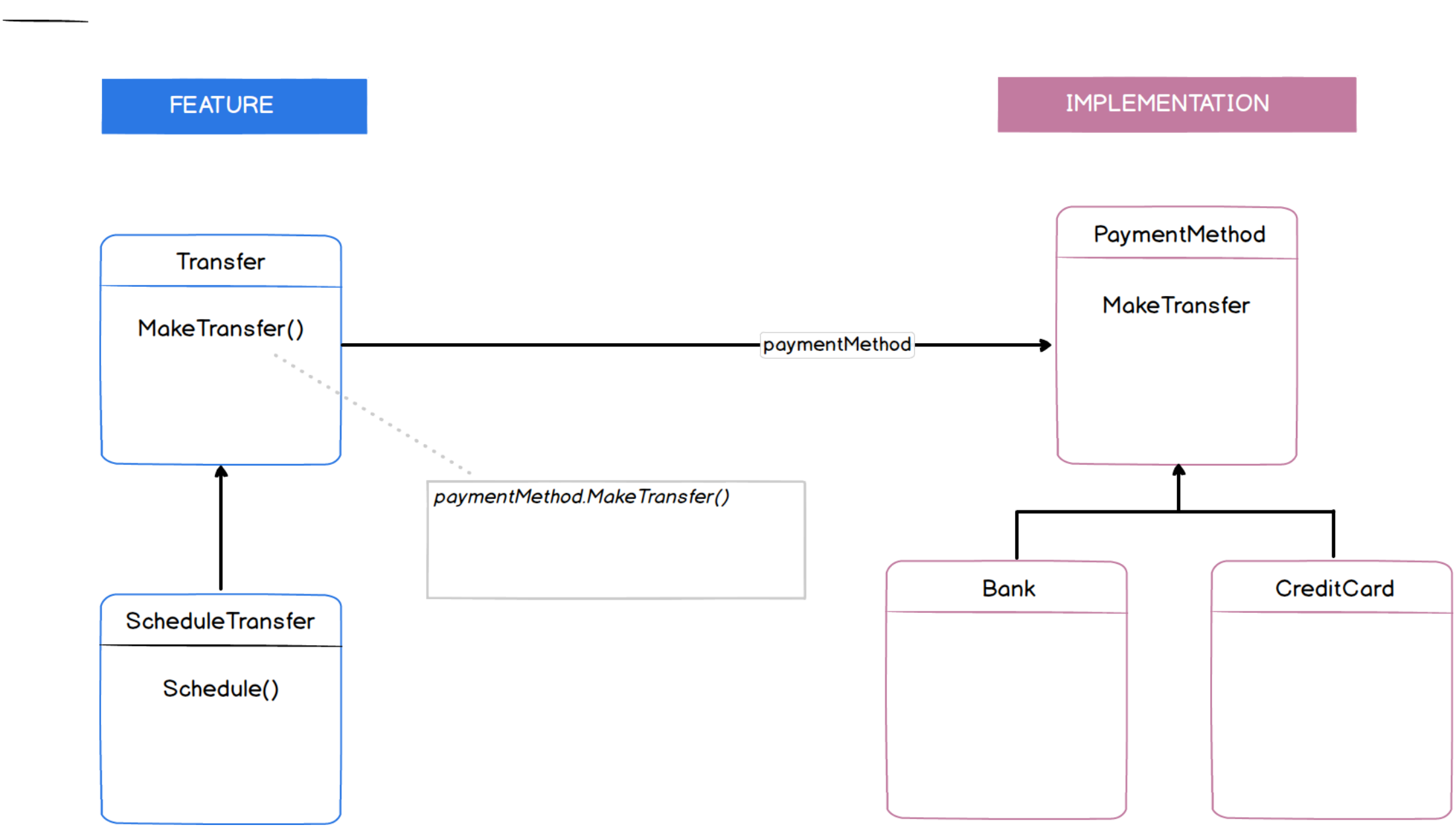
Object Adapter




Transfer
ScheduleTransfer
BankTransfer
CreditCardTransfer
BankScheduleTransfer
CreditCardScheduleTra

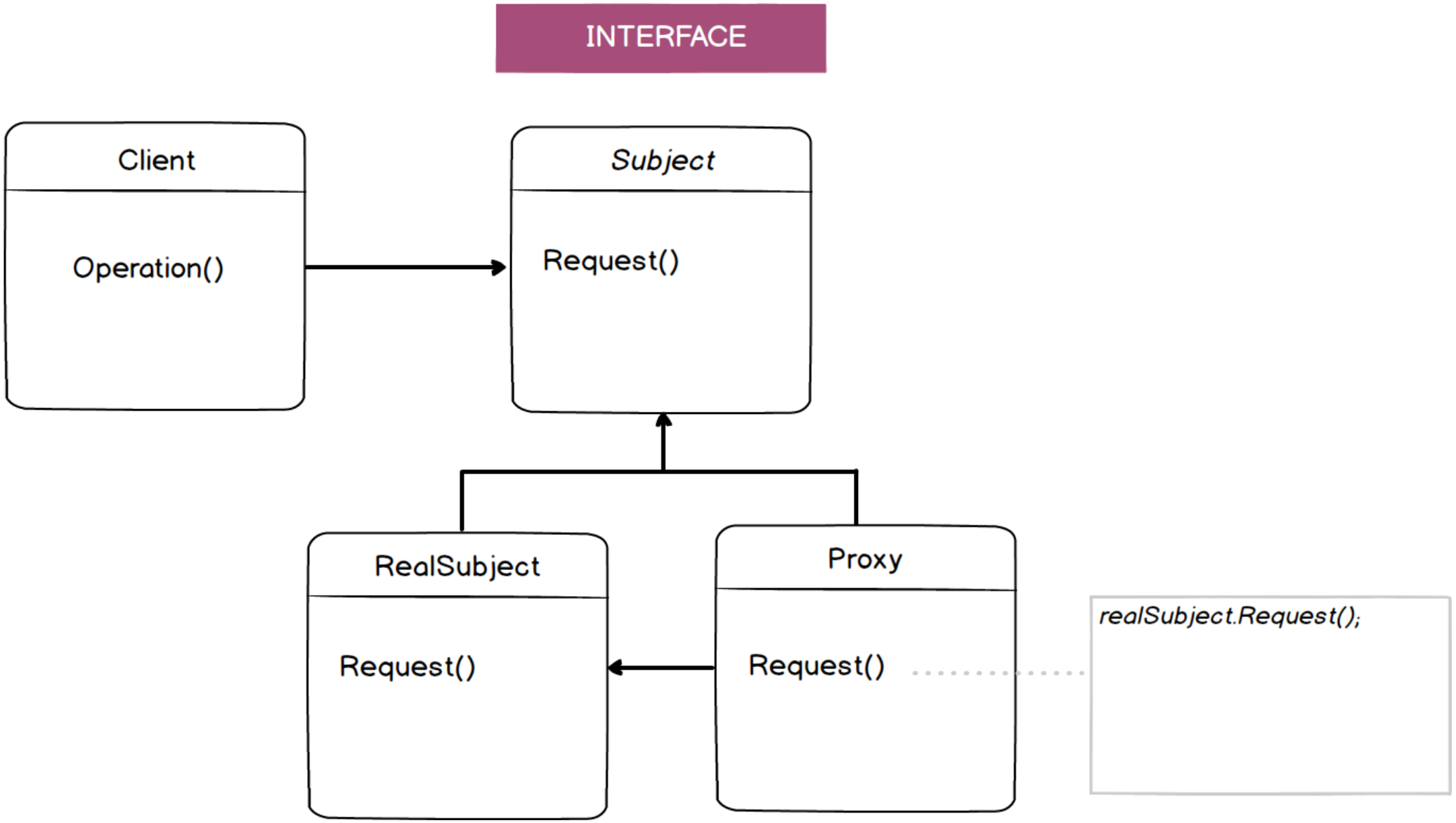

FEATURE
Transfer
ScheduleTransfer
BankTransfer
CreditCardTransfer
BankScheduleTransfer
CreditCardScheduleTransfer
IMPLEMENTATION


FEATURE
IMPLEMENTATION
Transfer
MakeTransfer()
paymentMethod
PaymentMethod
MakeTransfer
paymentMethod.MakeTransfer()
ScheduleTransfer
Schedule()
Bank
CreditCard


FEATURE
IMPLEMENTATION
Transfer
MakeTransfer()
paymentMethod
PaymentMethod
MakeTransfer
paymentMethod.MakeTransfer()
ScheduleTransfer
Schedule()
Bank
CreditCard

Proxy Pattern

Pozwala na stworzenie obiektu **zastępczego**
w miejsce innego obiektu.


INTERFACE
Client
Operation()
Subject
Request()
RealSubject
Request()
Proxy
Request()
realSubject.Request();

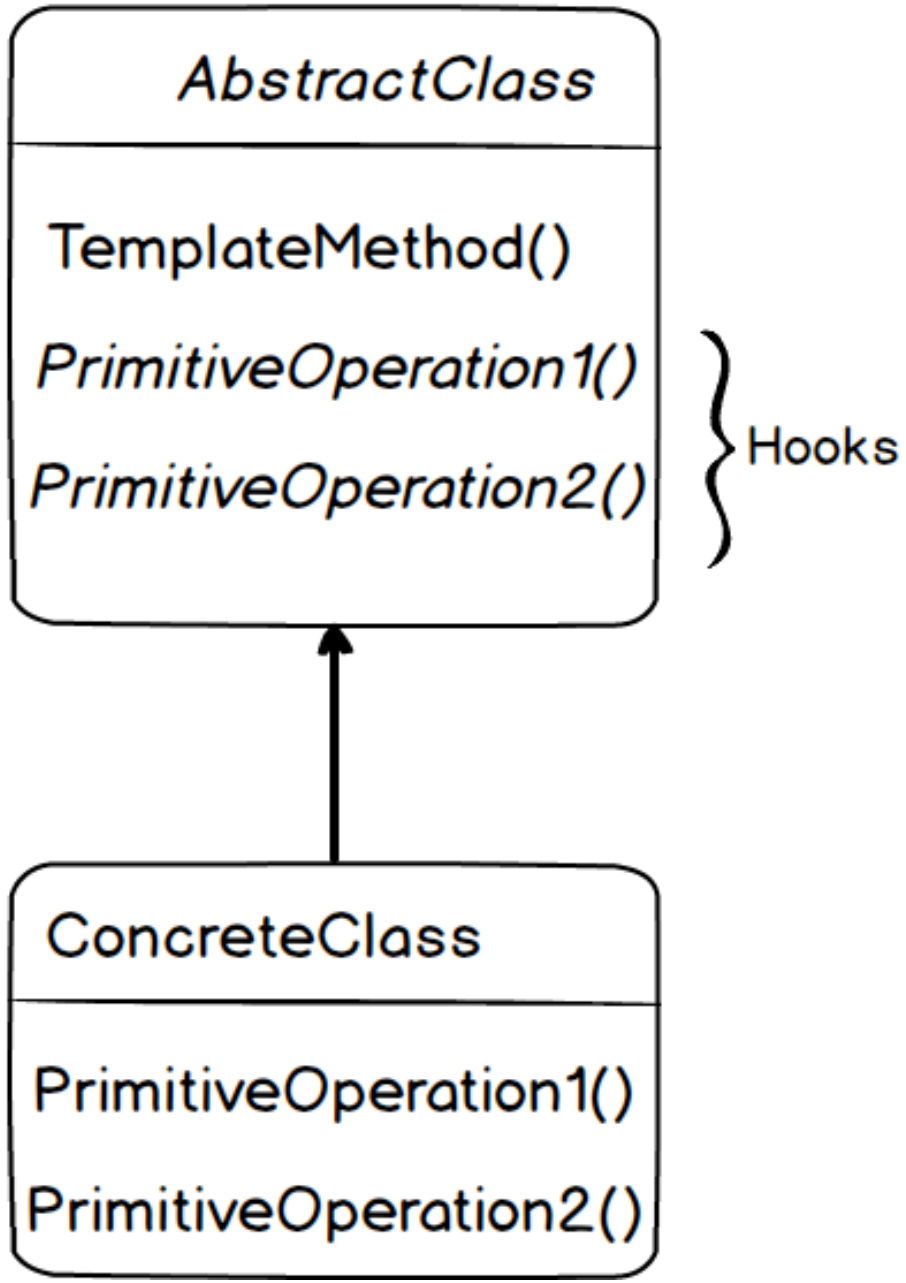
Decorator Pattern

Przedkładaj **kompozycję** nad dziedziczenie

WZORCE CZYNNOŚCIOWE

Template Method

Metoda, która jest **szkieletem** algorytmu,
a jego dokładna implementacja jest w klasach pochodnych.


AbstractClass
TemplateMethod()
PrimitiveOperation1()
PrimitiveOperation2()
Hooks
ConcreteClass
PrimitiveOperation1()
PrimitiveOperation2()

Strategy Pattern

Definiuje rodzinę **wymiennych** algorytmów.

Fasada Pattern

Ukrywa złożony zestaw klas
w **uproszczony** interfejs.

Command Pattern

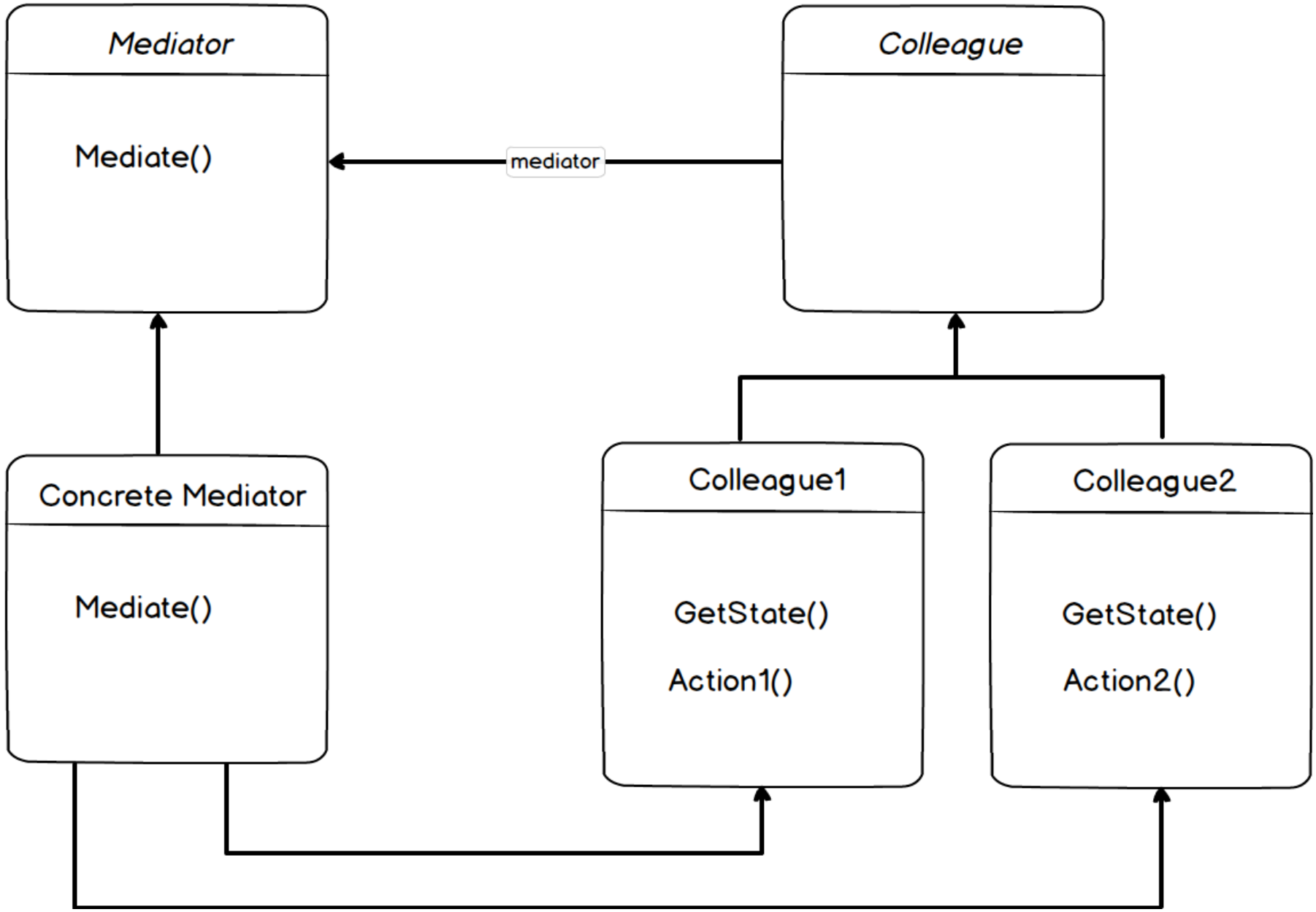
Traktujący żądanie wykonania czynności jako **obiekt**.

Chain of Responsibility Pattern

Pozwala przekazywać żądania wzdłuż
łańcucha obiektów obsługujących

Mediator Pattern

Zapewnia **jednolity interfejs** do różnych elementów danego podsystemu


Mediator
Mediate()
mediator
Colleague
Concrete Mediator
Mediate()
Colleague1
GetState()
Action1()
Colleague2
GetState()
Action2()

Interpreter Pattern

Umożliwia zdefiniowanie opisu gramatyki
języka interpretowalnego

Composite Pattern

Komponuje obiekty w
struktury drzewiaste

Iterator Pattern

Umożliwia dostęp do podobiektów
zgrupowanych w większym obiekcie

State Pattern

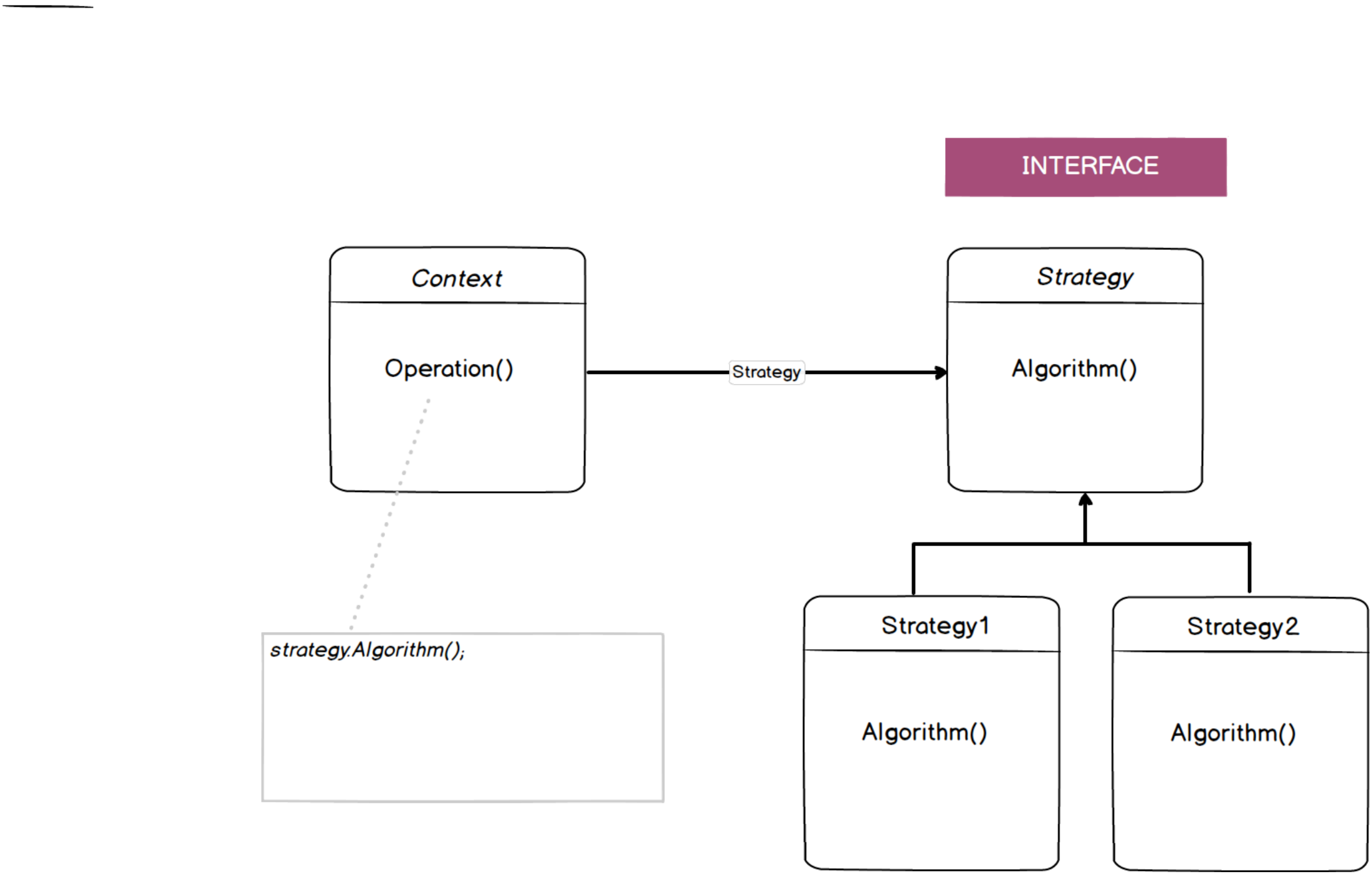
Pozwala obiektować zmieniać swoje zachowanie
gdy zmieni się jego **stan wewnętrzny**

Memento Pattern

Umożliwia **zapamiętanie** stanu obiektu i
przywrócenie zapamiętanego stanu obiektu

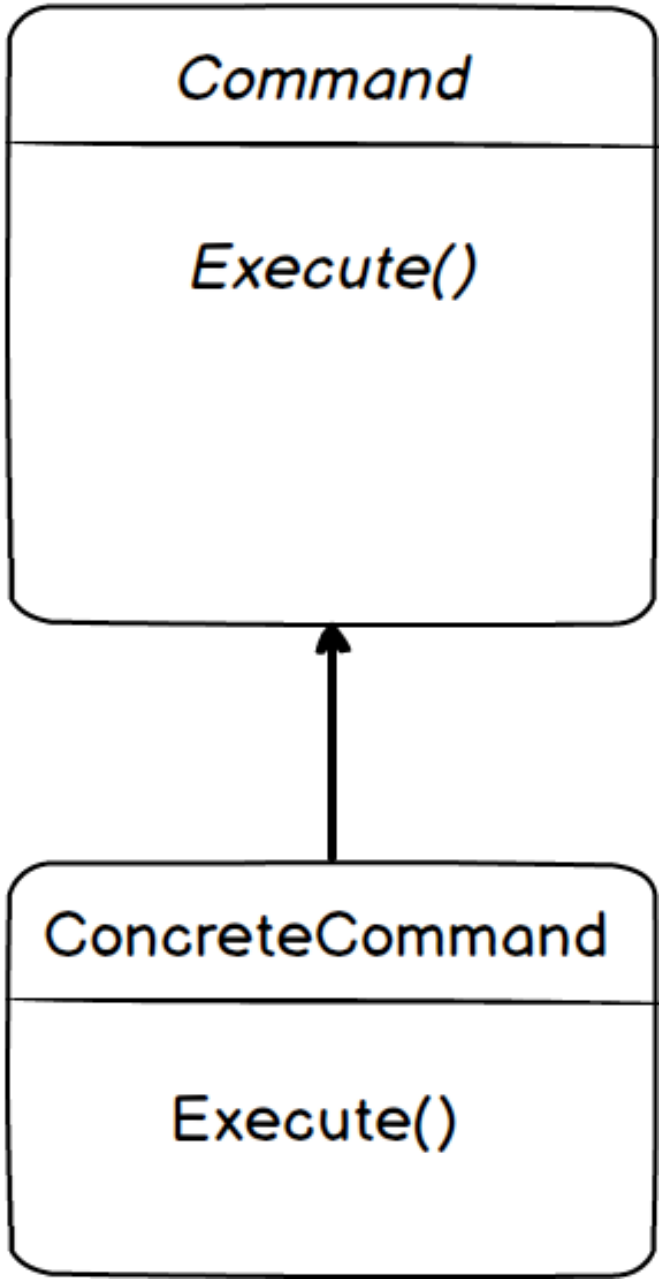
Null Object Pattern

Umożliwia uniknięcie sprawdzenia,
czy wartość jest różna od **null**


INTERFACE
Context
Operation()
Strategy
Strategy
Algorithm()
strategy.Algorithm();
Strategy1
Algorithm()
Strategy2
Algorithm()

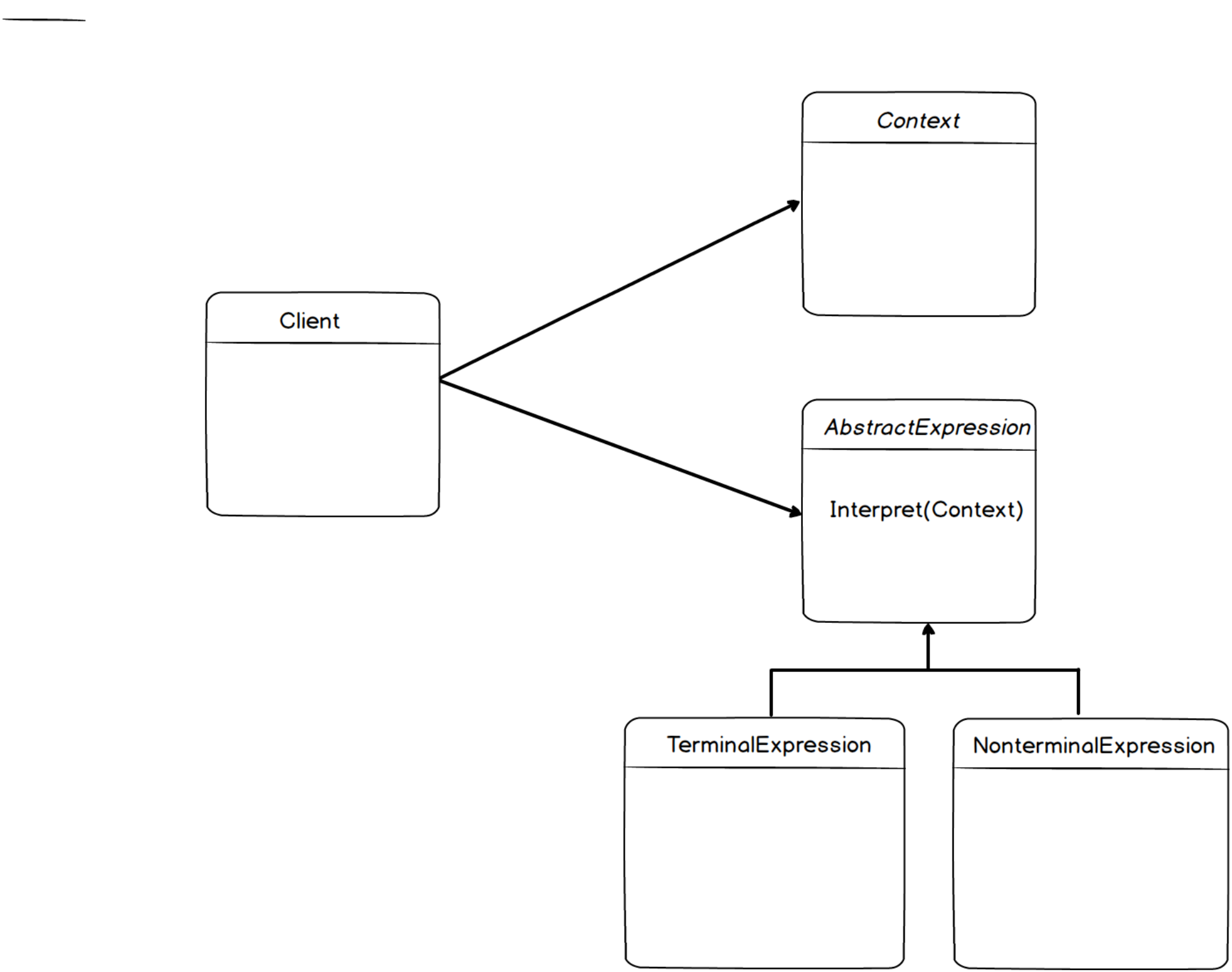
Observer Pattern

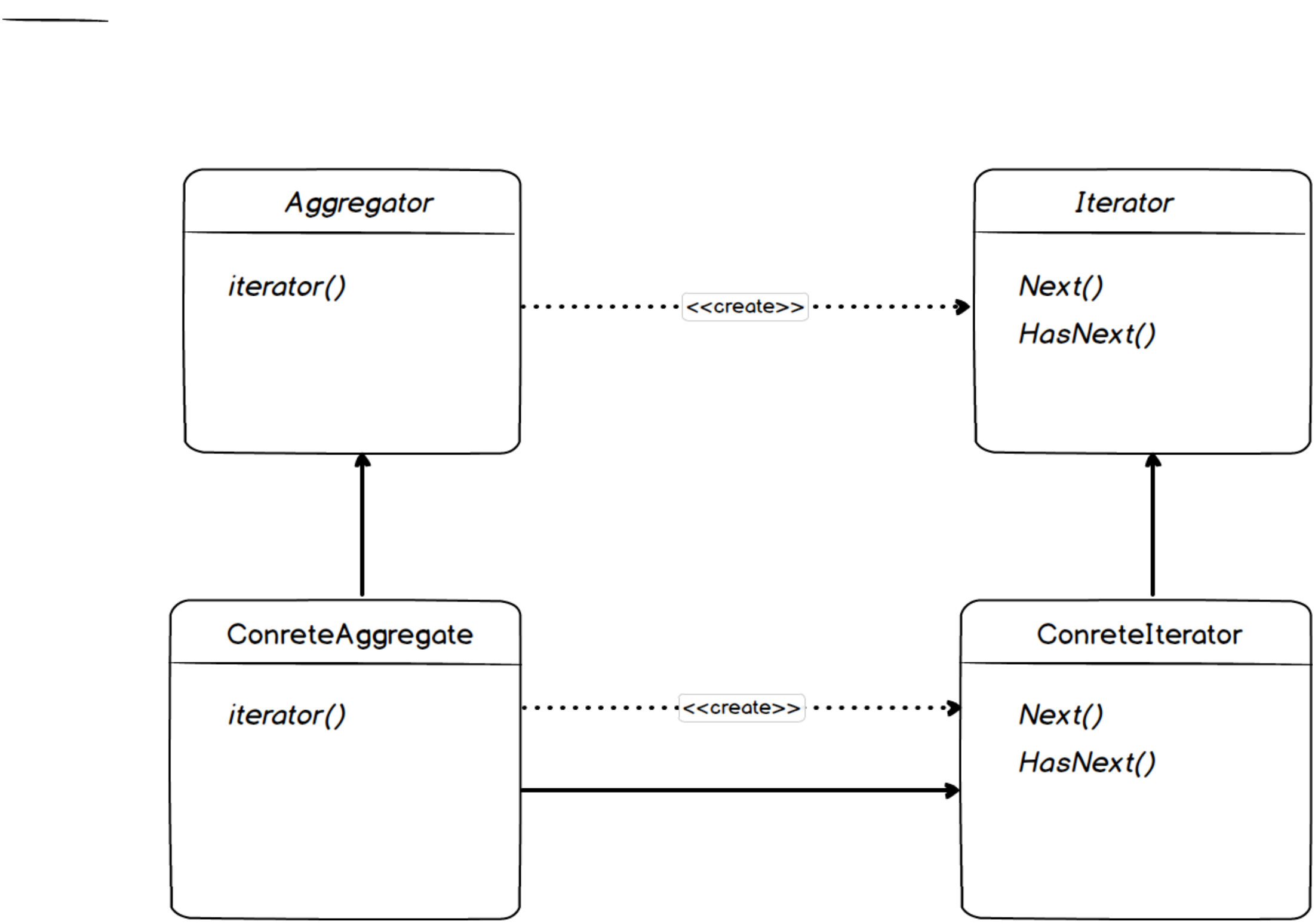
Powiadamiania zainteresowane obiekty
o zmianie stanu innego obiektu

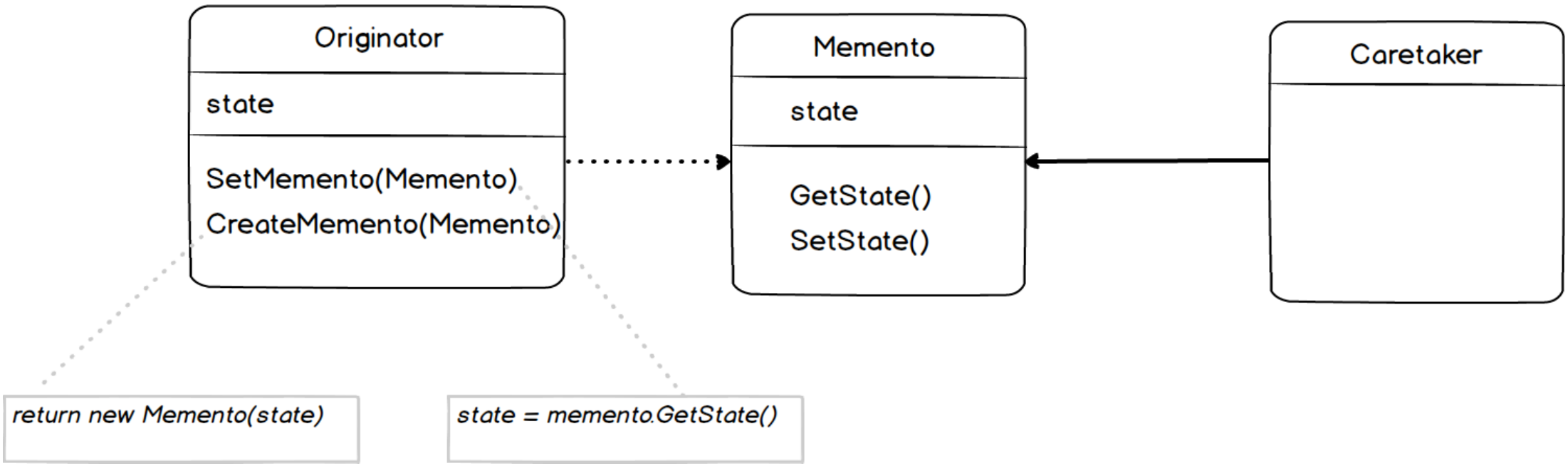

Command
Execute()
ConcreteCommand
Execute()

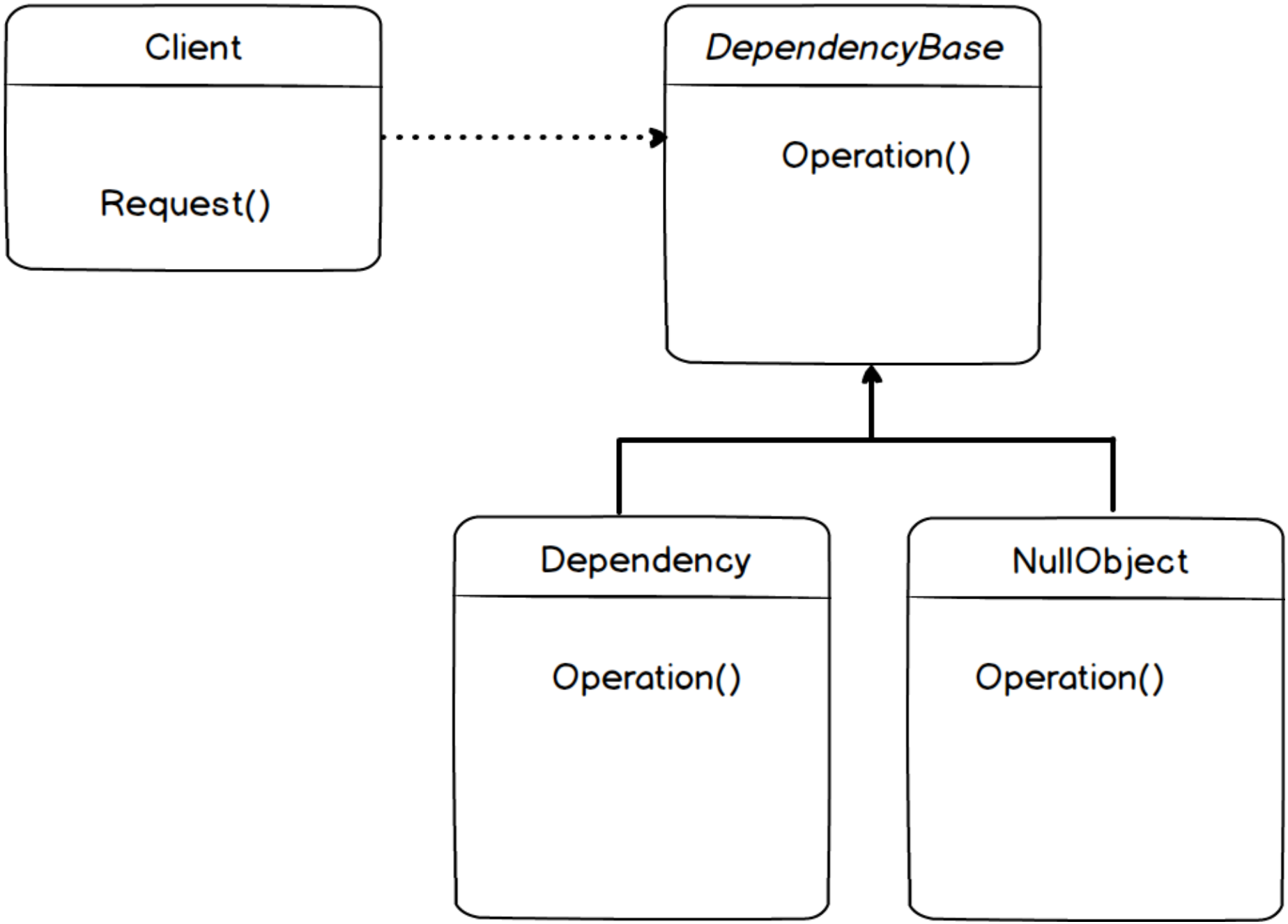
Visitor Pattern

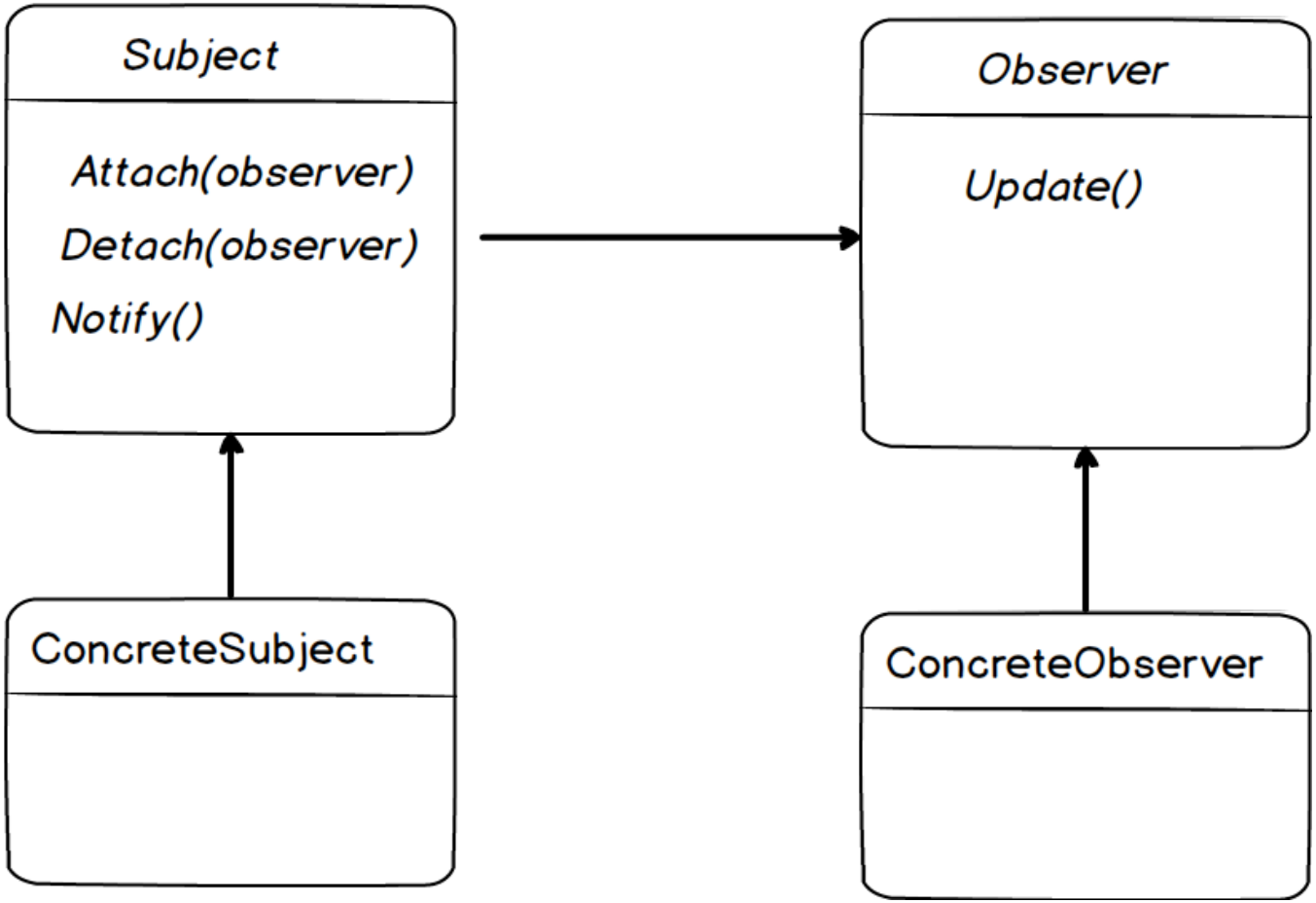
Umożliwia **odseparowanie** algorytmu od struktury obiektowej na której operuje

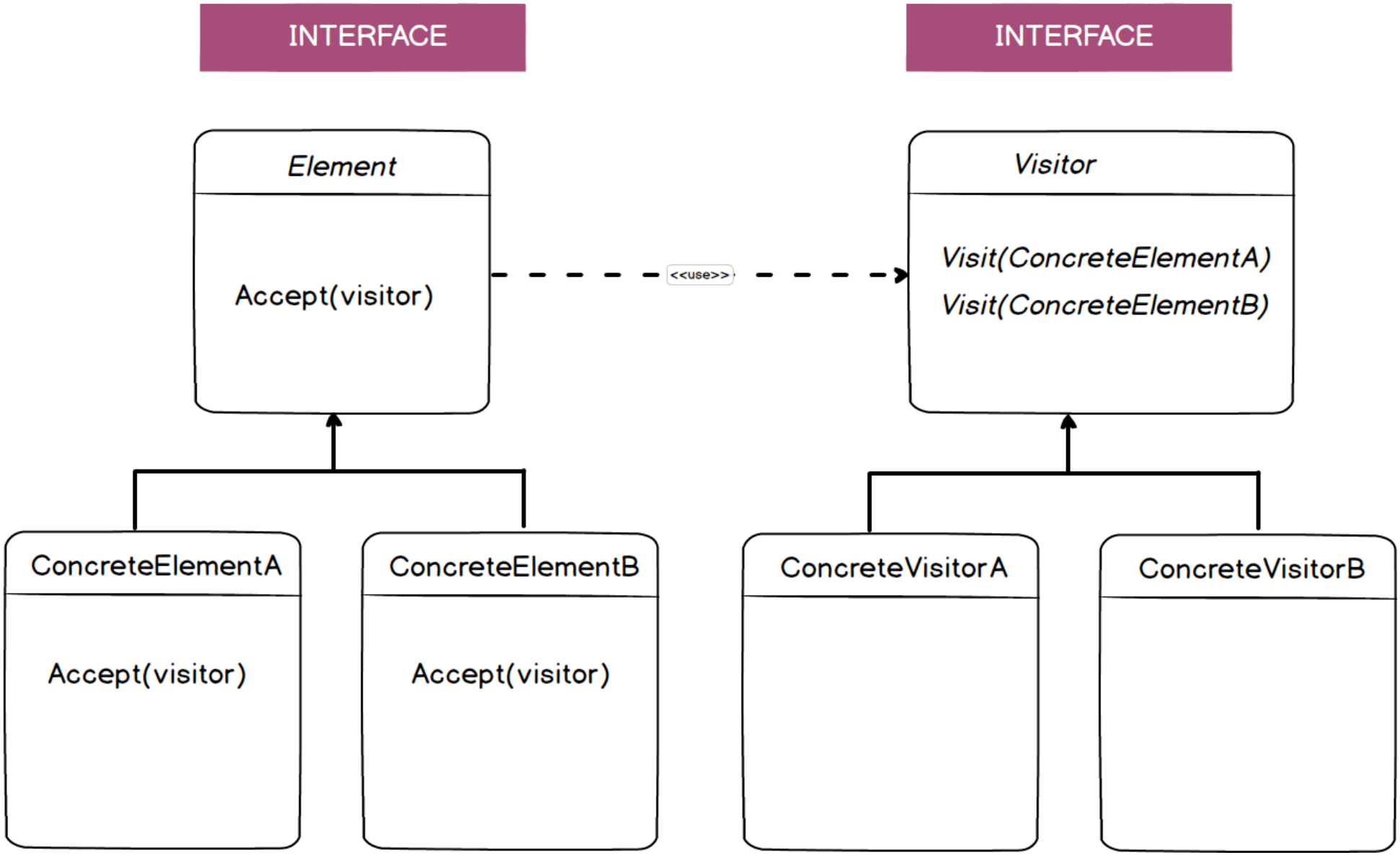

Context
Client
AbstractExpression
Interpret(Context)
TerminalExpression
NonterminalExpression


Aggregator
iterator()
<<create>>
Iterator
Next()
HasNext()
ConreteAggregate
iterator()
<<create>>
ConreteIterator
Next()
HasNext()


Originator
state
SetMemento(Memento)
CreateMemento(Memento)
Memento
state
GetState()
SetState()
Caretaker
return new Memento(state)
state = memento.GetState()

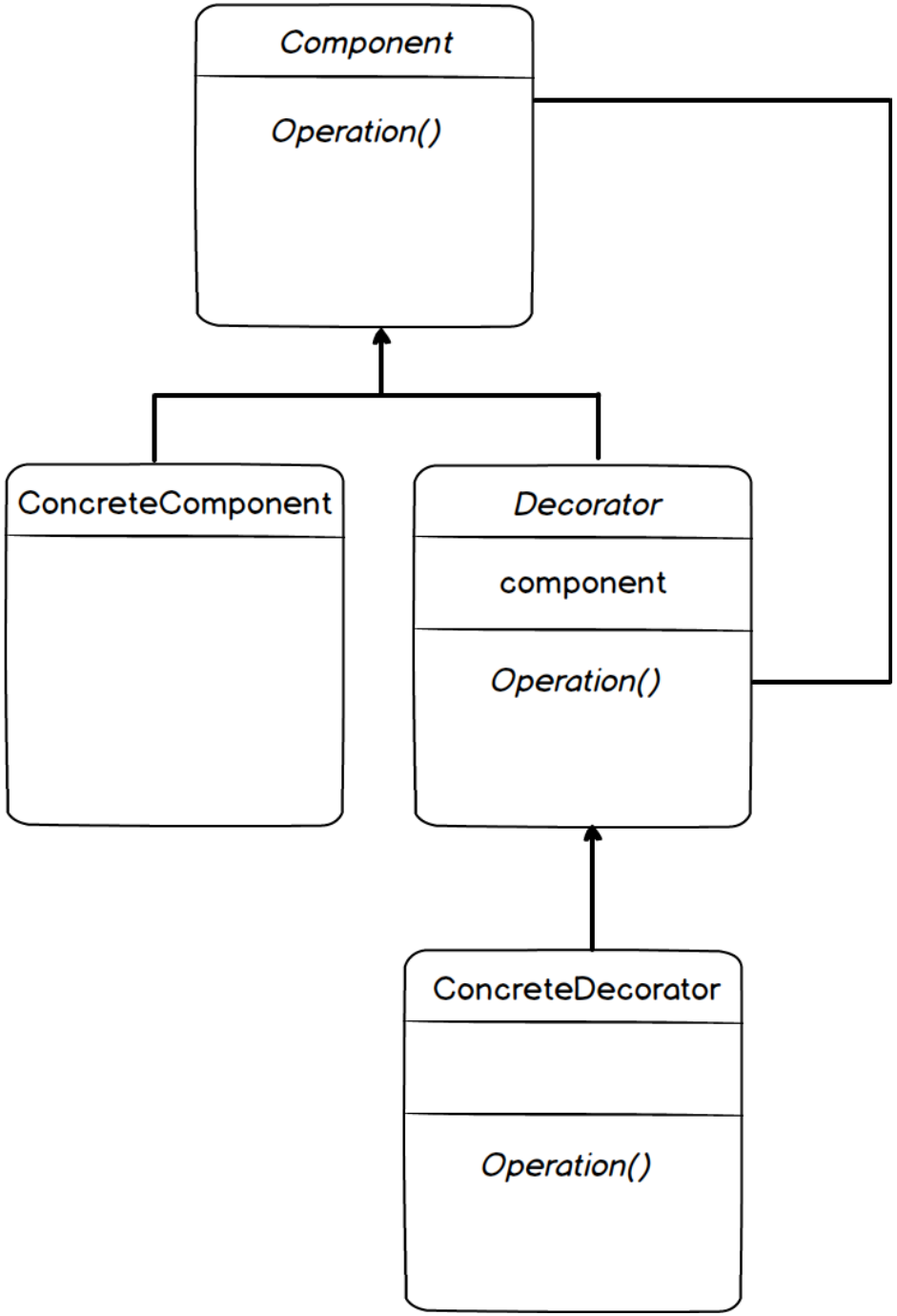

Client
Request()
DependencyBase
Operation()
Dependency
Operation()
NullObject
Operation()

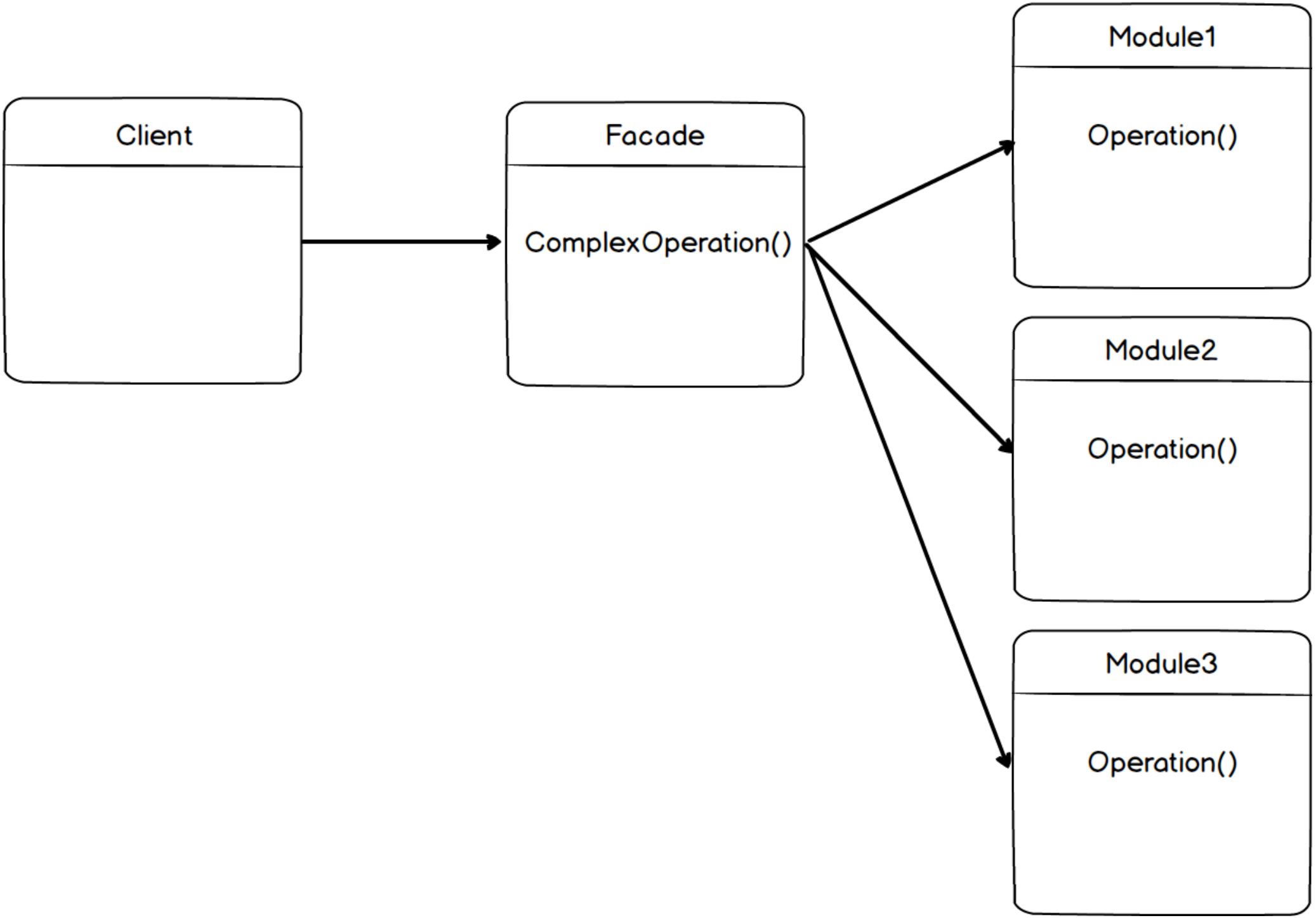

Subject
Attach(observer)
Detach(observer)
Notify()
Observer
Update()
ConcreteSubject
ConcreteObserver

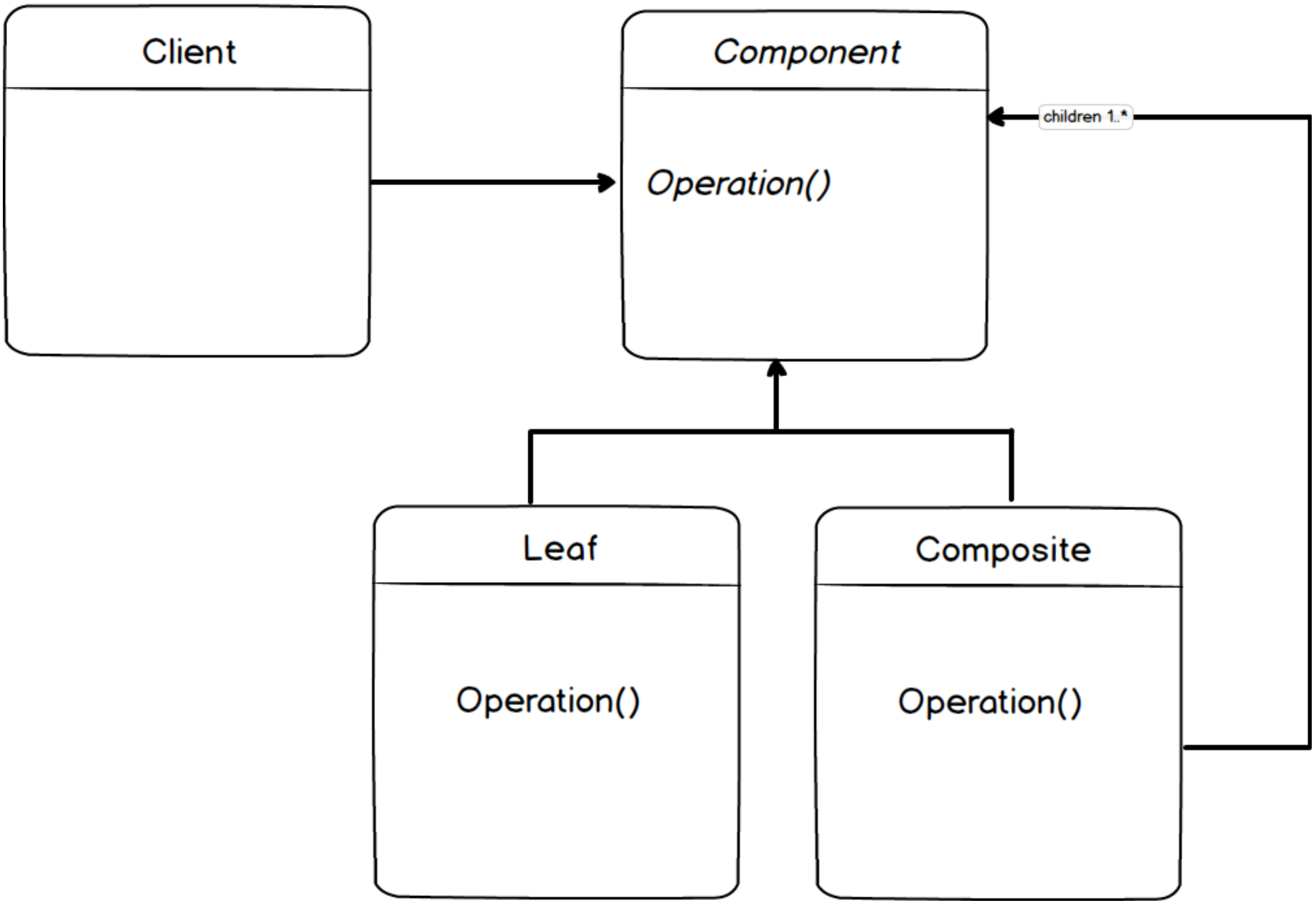

INTERFACE
INTERFACE
Element
Accept(visitor)
<<use>>
Visitor
Visit(ConcreteElementA)
Visit(ConcreteElementB)
ConcreteElementA
Accept(visitor)
ConcreteElementB
Accept(visitor)
ConcreteVisitorA
ConcreteVisitorB

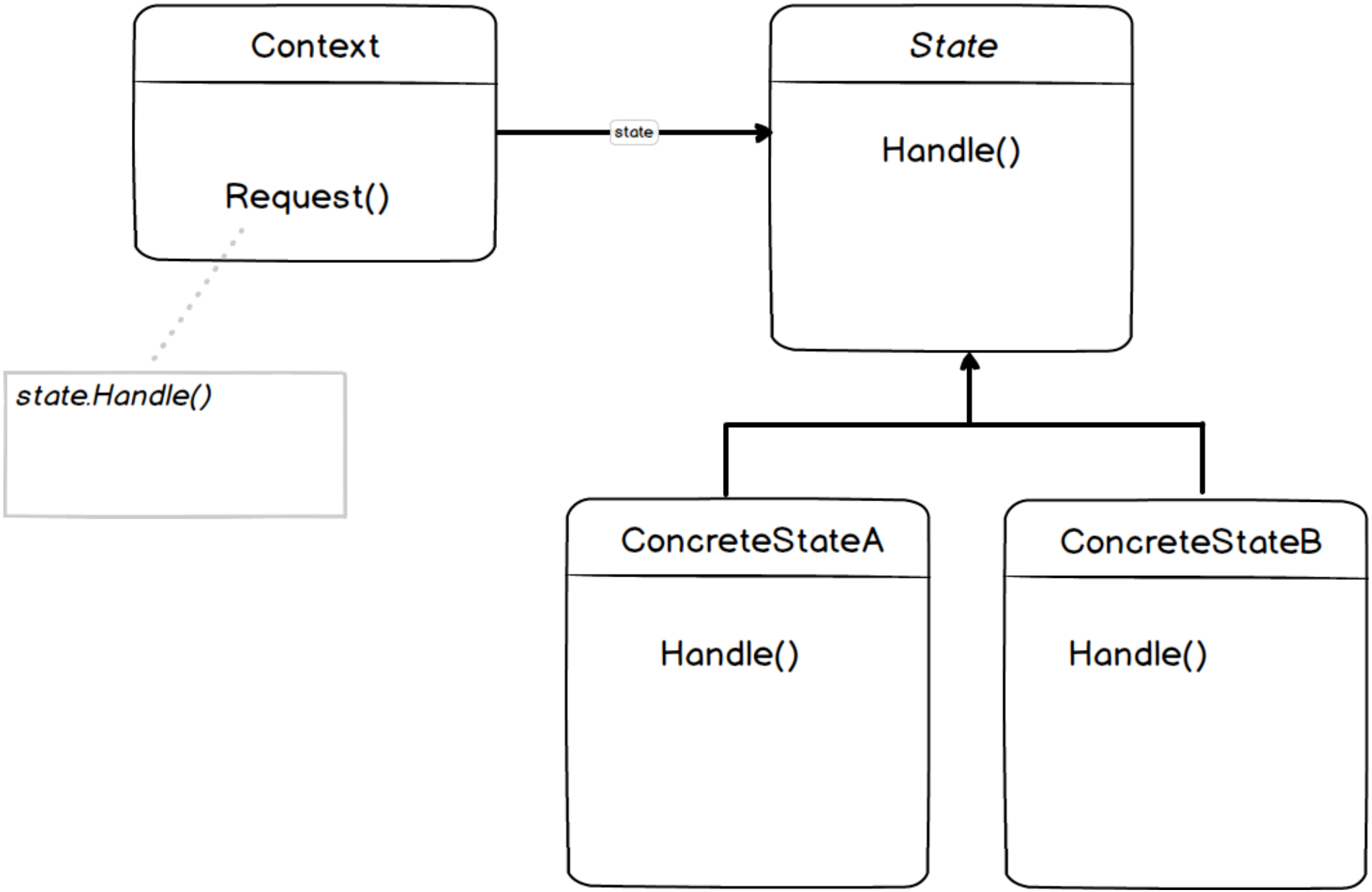
Flyweight Pattern

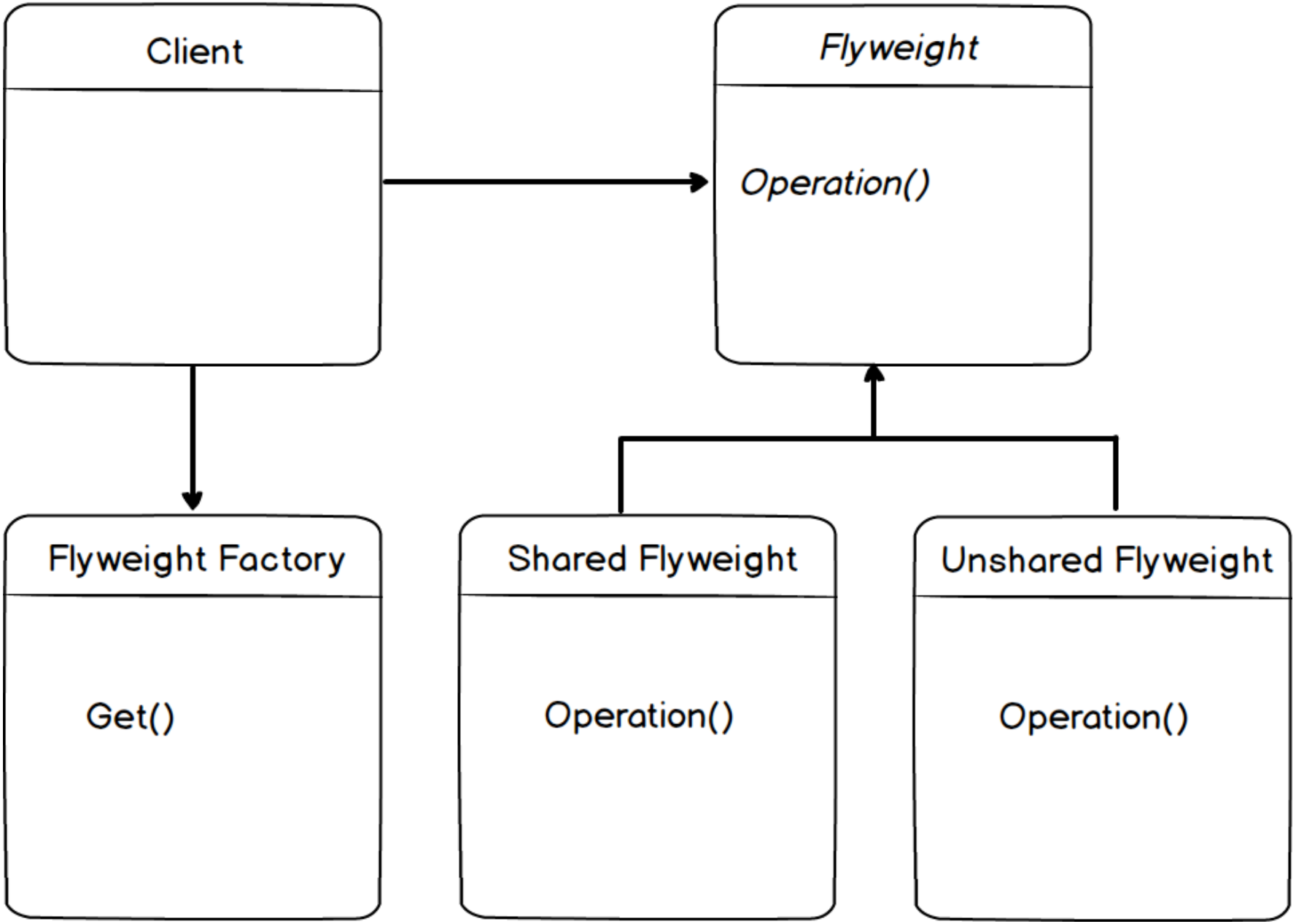
Pozwala zmieścić więcej obiektów w pamięci
poprzez współdzielenie części opisu ich stanów


Component
Operation()
ConcreteComponent
Decorator
component
Operation()
ConcreteDecorator
Operation()


Client
Facade
ComplexOperation()
Module1
Operation()
Module2
Operation()
Module3
Operation()


Client
Component
Operation()
children 1..*
Leaf
Operation()
Composite
Operation()


Context
Request()
state
State
Handle()
state.Handle()
ConcreteStateA
Handle()
ConcreteStateB
Handle()


Client
Flyweight
Operation()
Flyweight Factory
Get()
Shared Flyweight
Operation()
Unshared Flyweight
Operation()