

# WZORCE PROJEKTOWE

WZORCE  
KREACYJNE

WZORCE  
STRUKTURALNE

WZORCE  
CZYNNOŚCIOWE

WZORCE KREACYJNE

# Singleton Pattern

Zapewnia, że klasa posiada  
pojedynczą instancję

---

## ConfigManager

Get()

Set()

GetInstance()

---

## ConfigManager

Get()

Set()

- ConfigManager()

---

## ConfigManager

Get()

Set()

- ConfigManager()

- instance

---

## ConfigManager

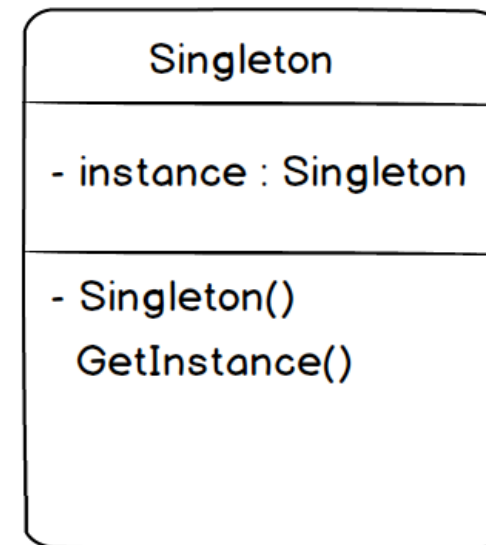
Get()

Set()

- ConfigManager()

- instance

GetInstance()

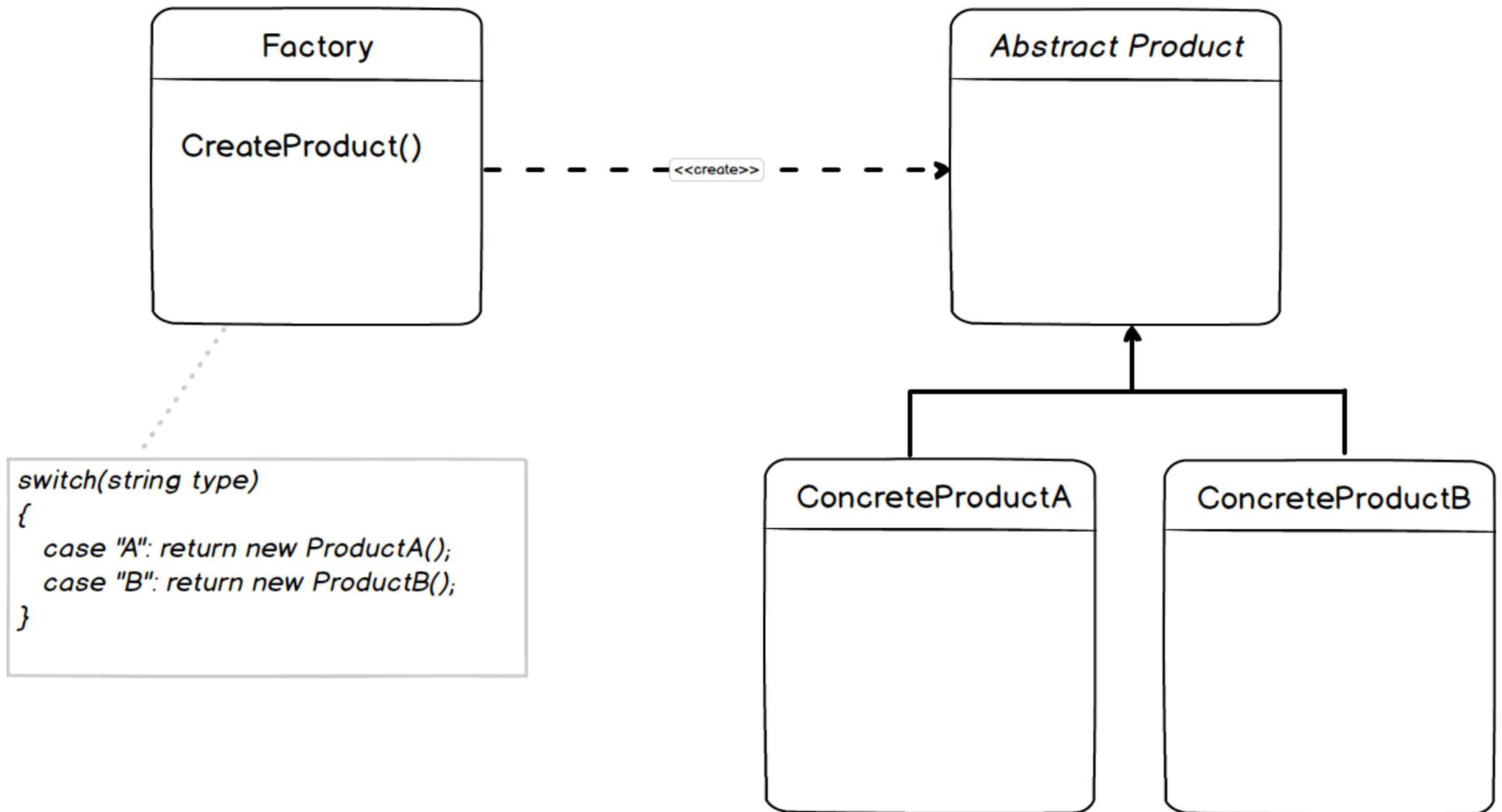


Pozwala zachować pewność, że istnieje wyłącznie **jedna instancja** danej klasy.



# Simple Factory Pattern

Zapewnia tworzenie obiektu na podstawie wielu warunków



# Abstract Factory Pattern

Zapewnia interfejs do tworzenia  
rodziny powiązanych obiektów

## INTERFACE

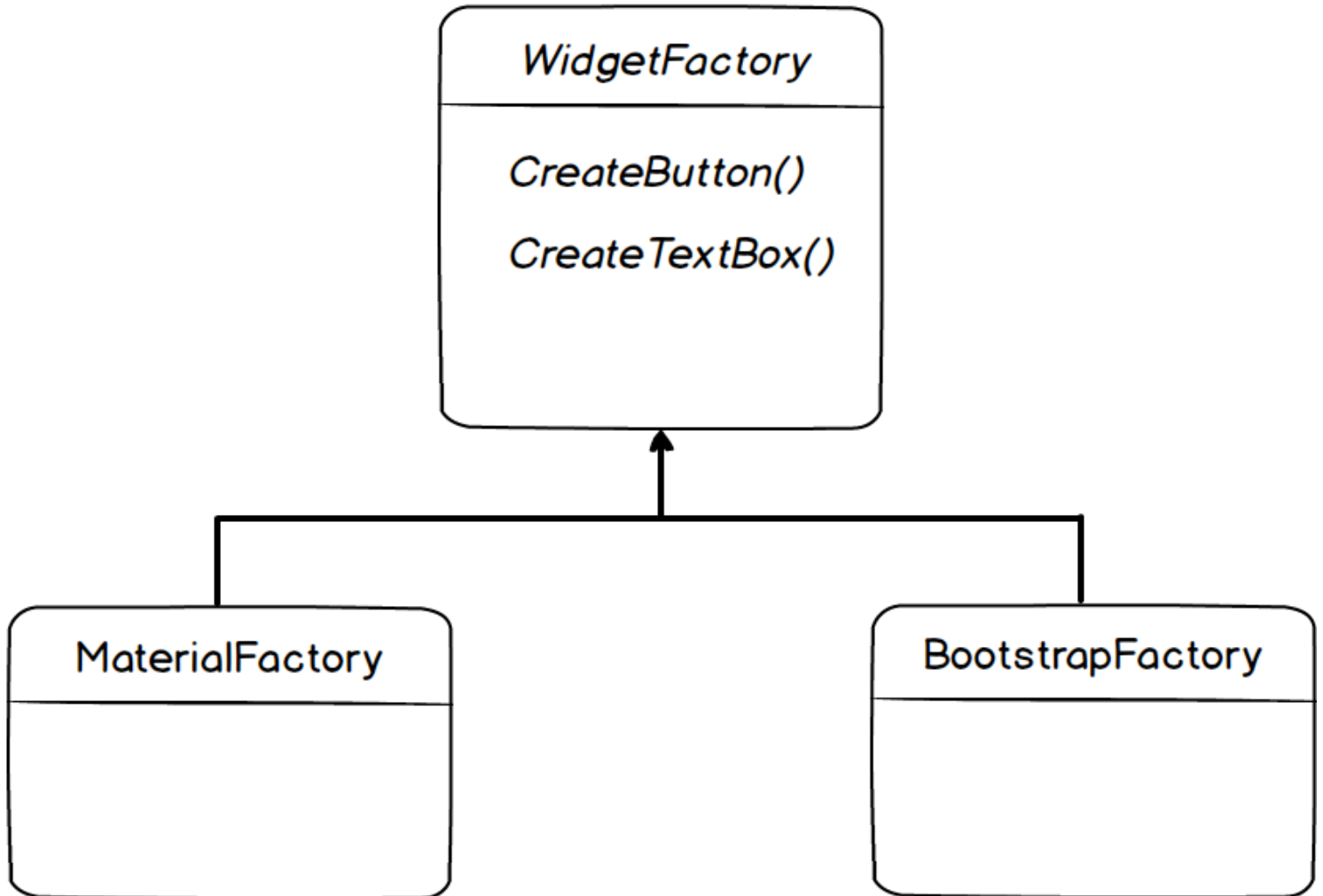
*WidgetFactory*

*CreateButton()*

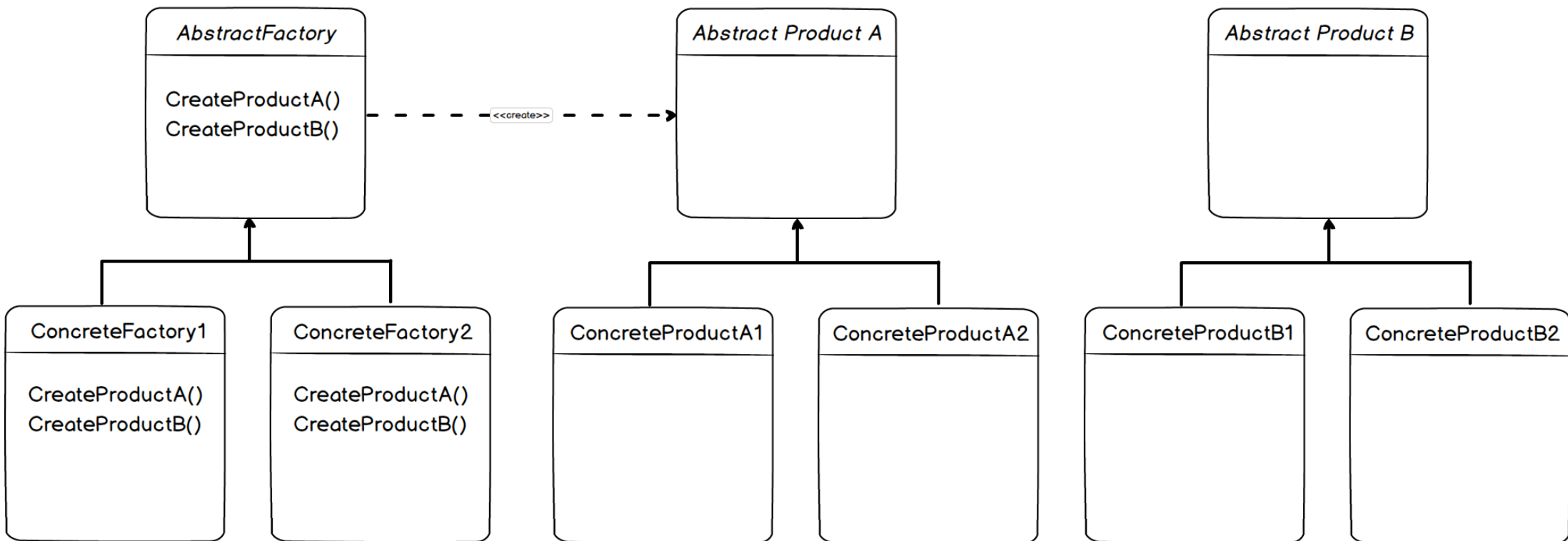
*CreateTextBox()*

MaterialFactory

BootstrapFactory



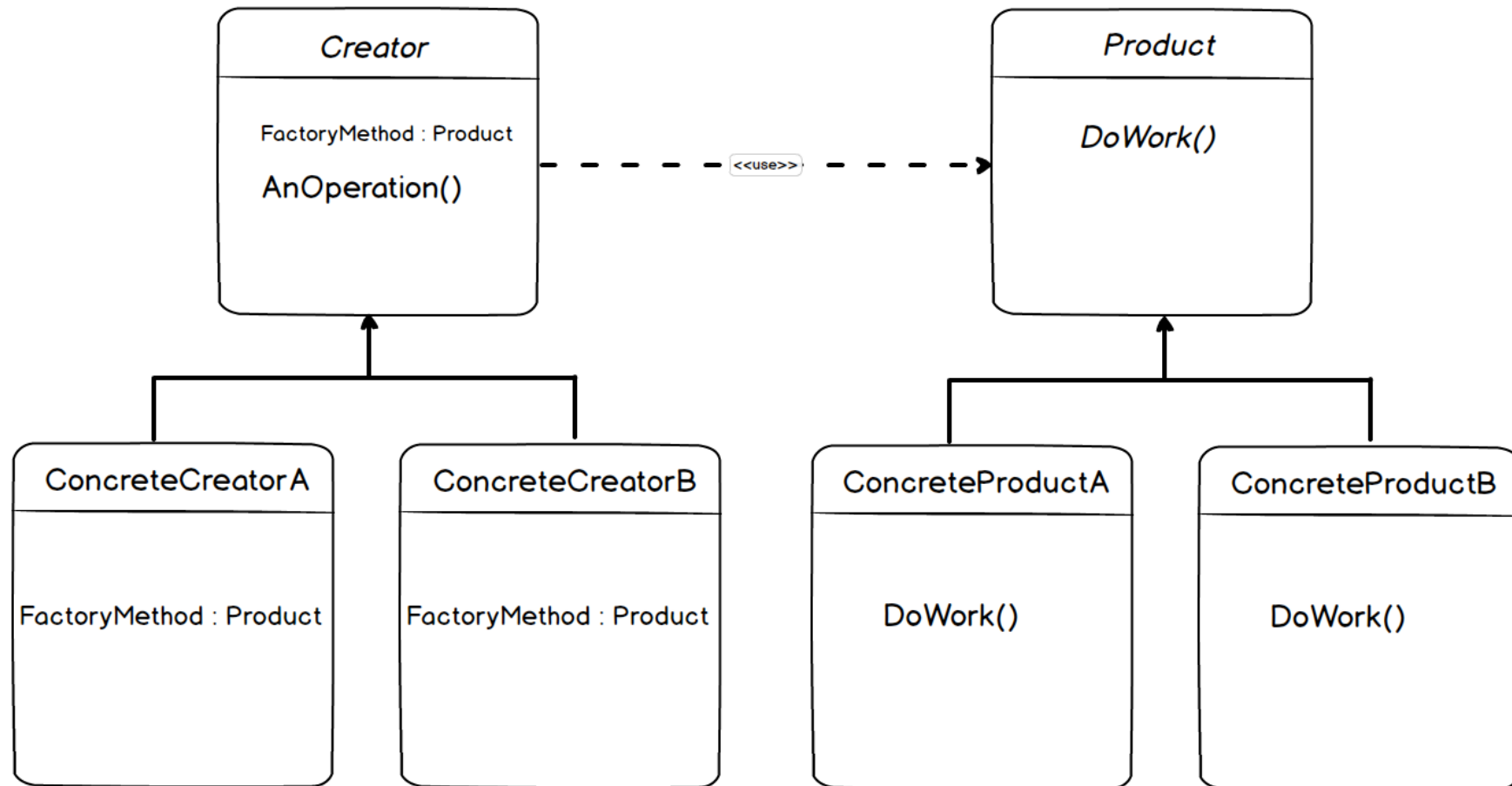
## INTERFACE



# Factory Method Pattern

Odkłada tworzenie obiektu  
do **podklas**

## INTERFACE

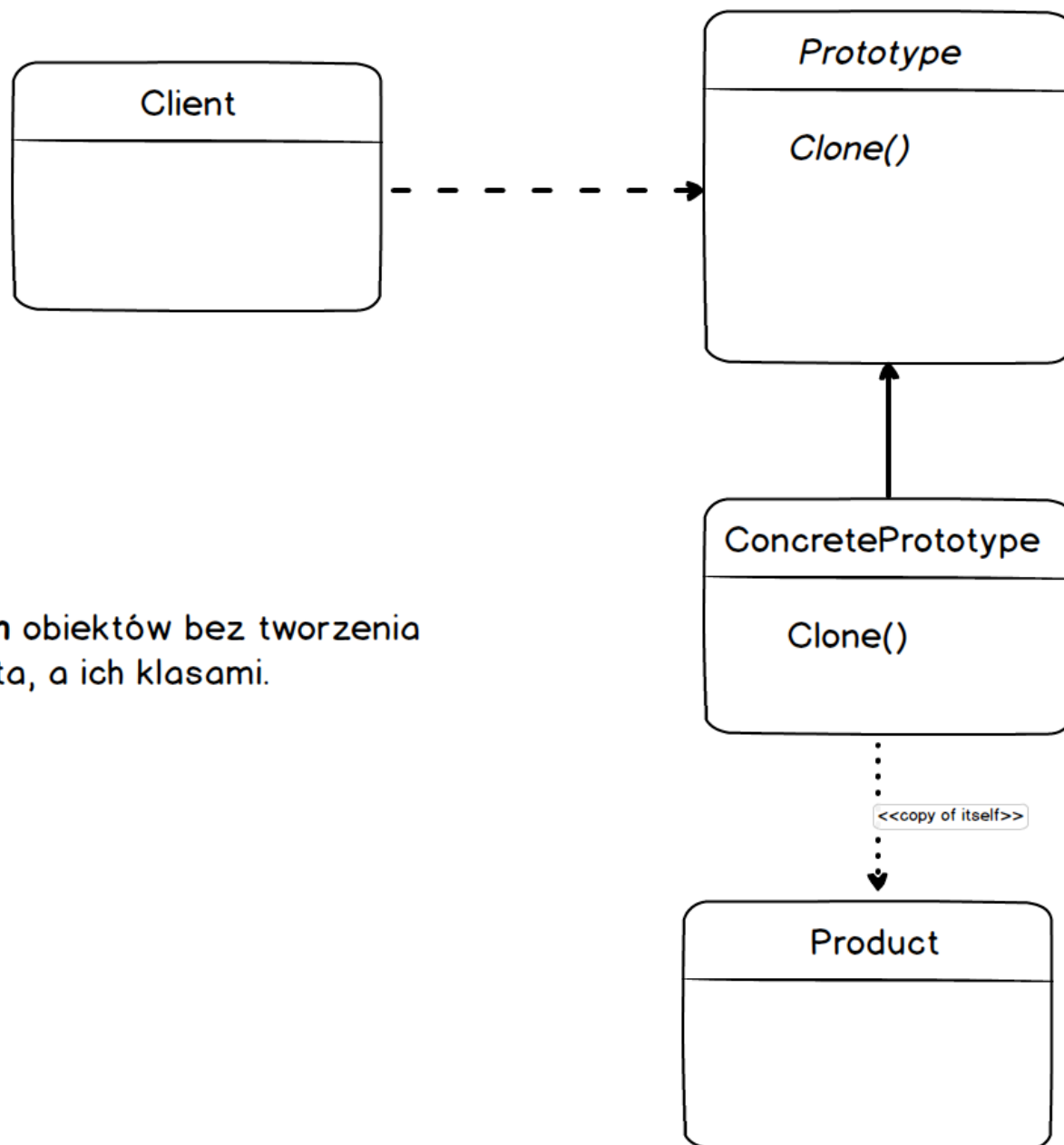


# Prototype Pattern

Tworzy nowe obiekty  
kopiując istniejący obiekt



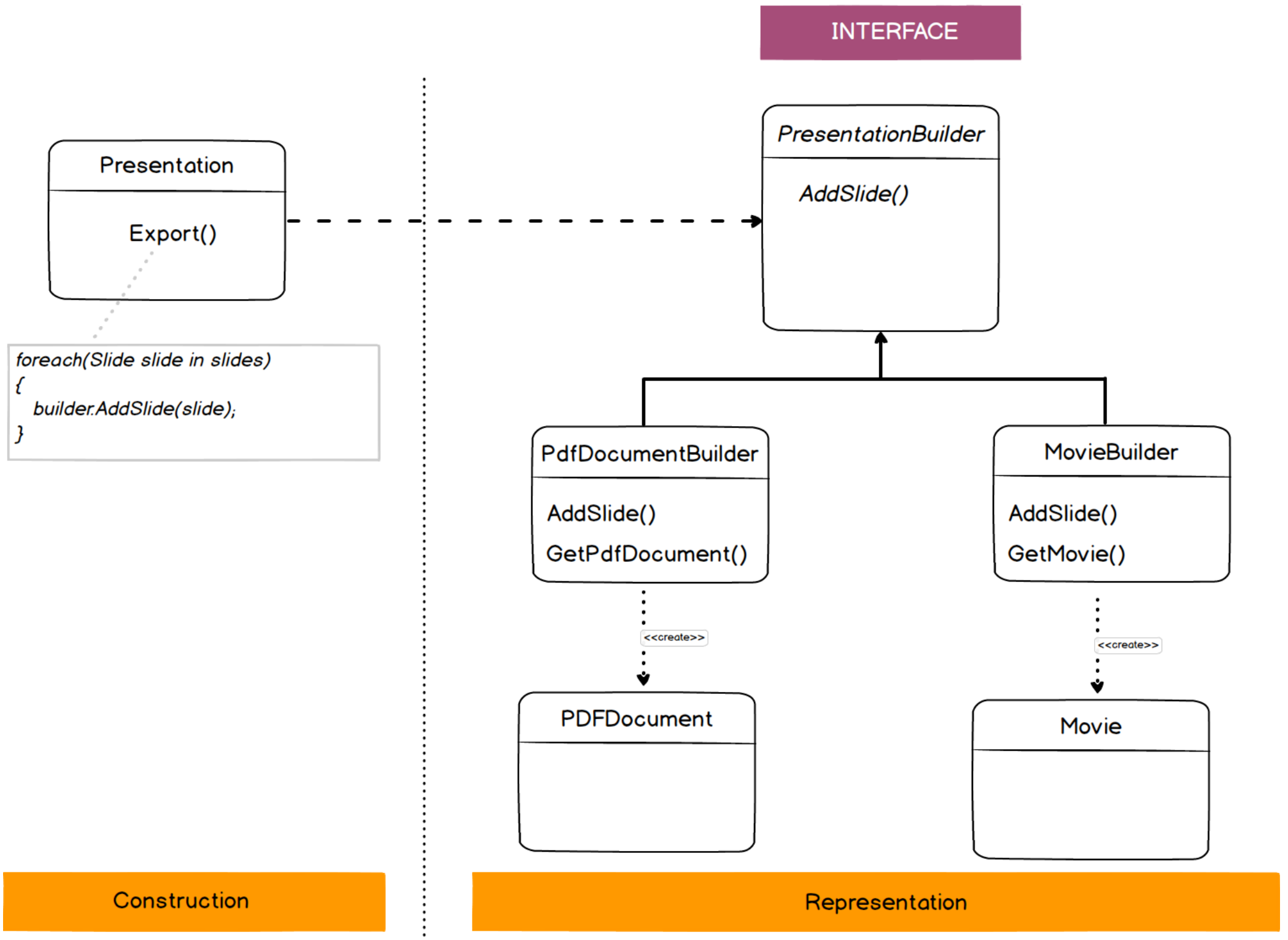
## INTERFACE



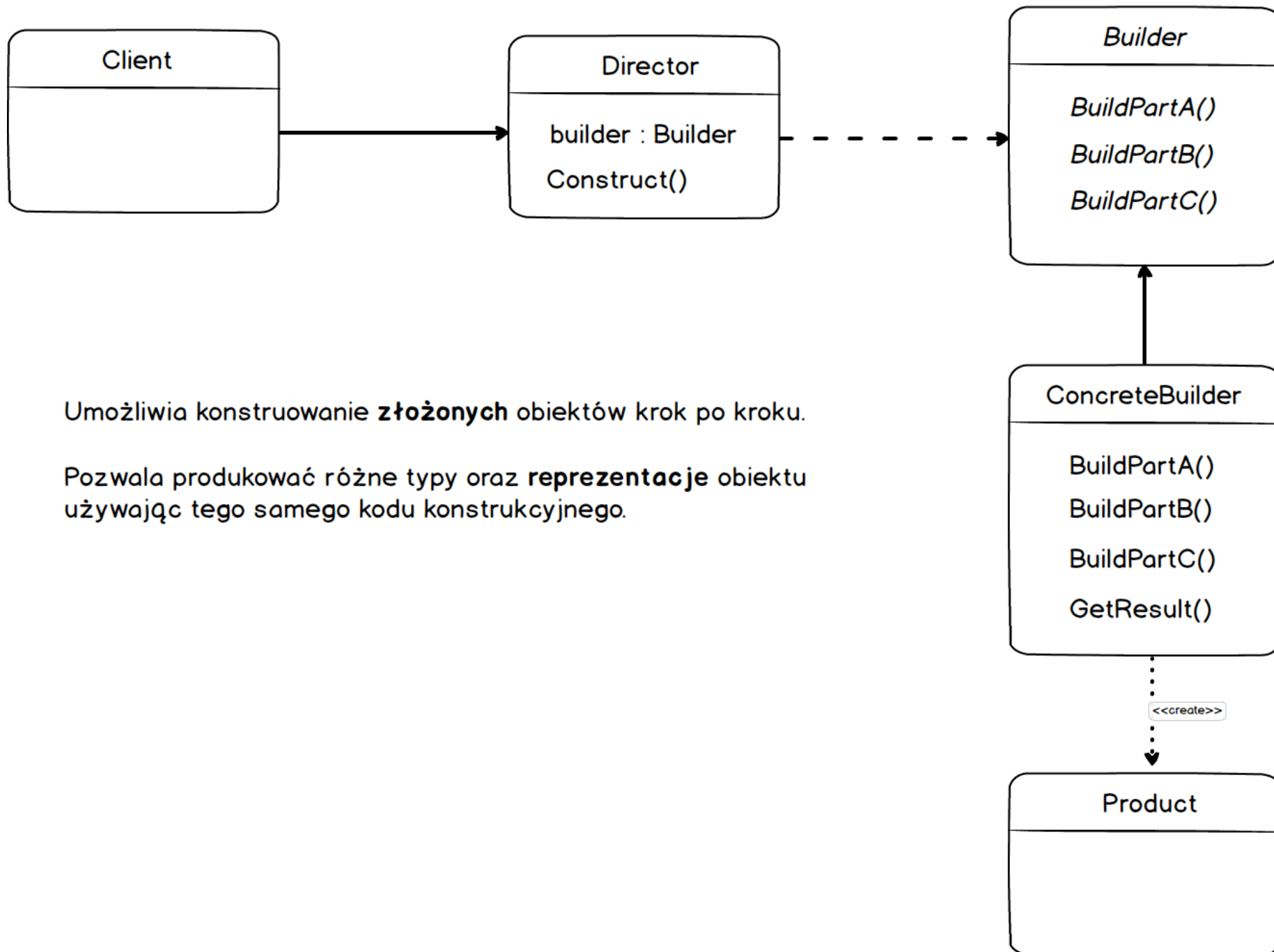
Umożliwia kopiowanie **istniejących** obiektów bez tworzenia zależności pomiędzy kodem klienta, a ich klasami.

# Builder Pattern

Oddziela konstrukcję obiektu  
od jego reprezentacji



## INTERFACE



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

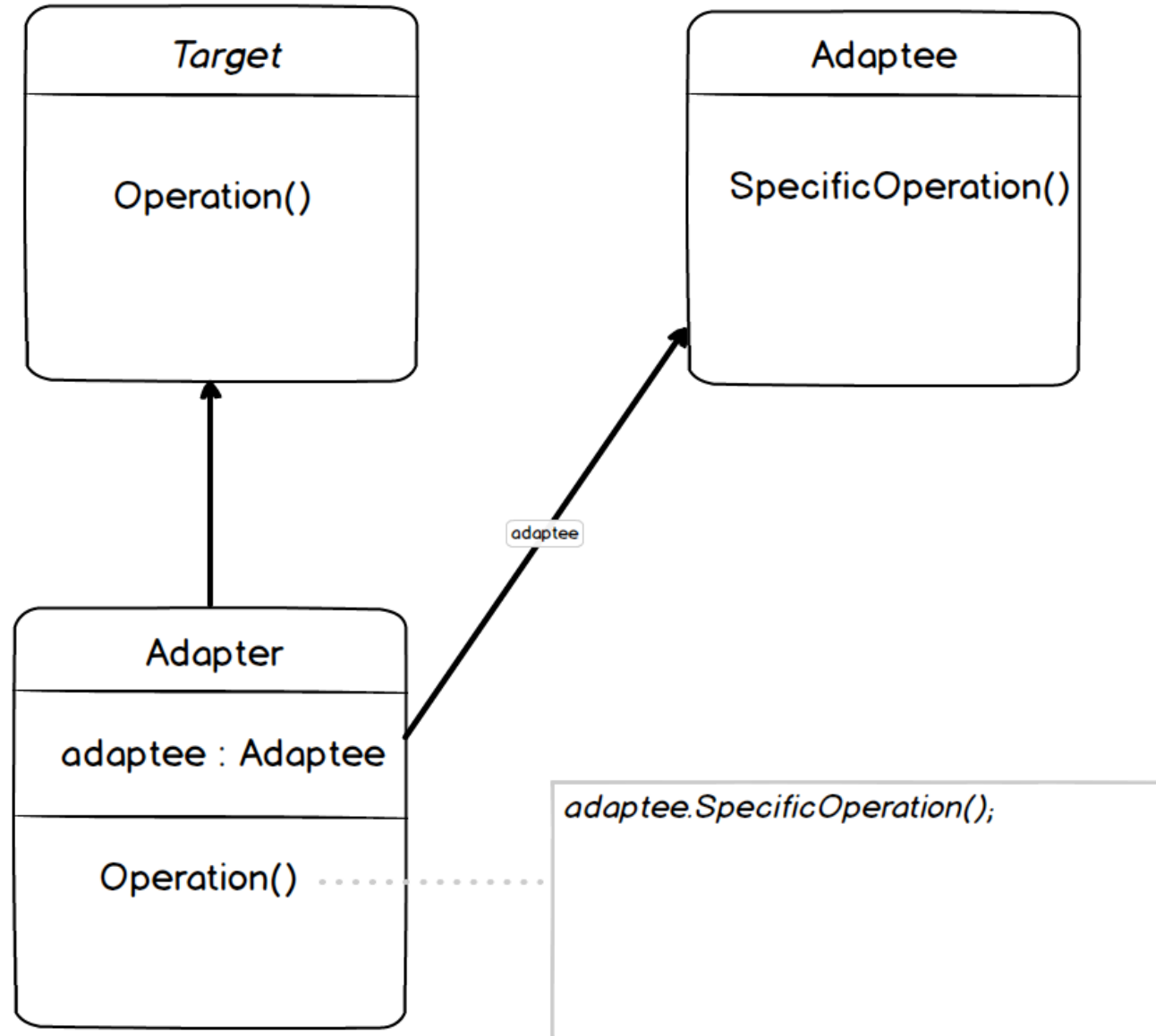
Pozwala produkować różne typy oraz **reprezentacje** obiektu używając tego samego kodu konstrukcyjnego.

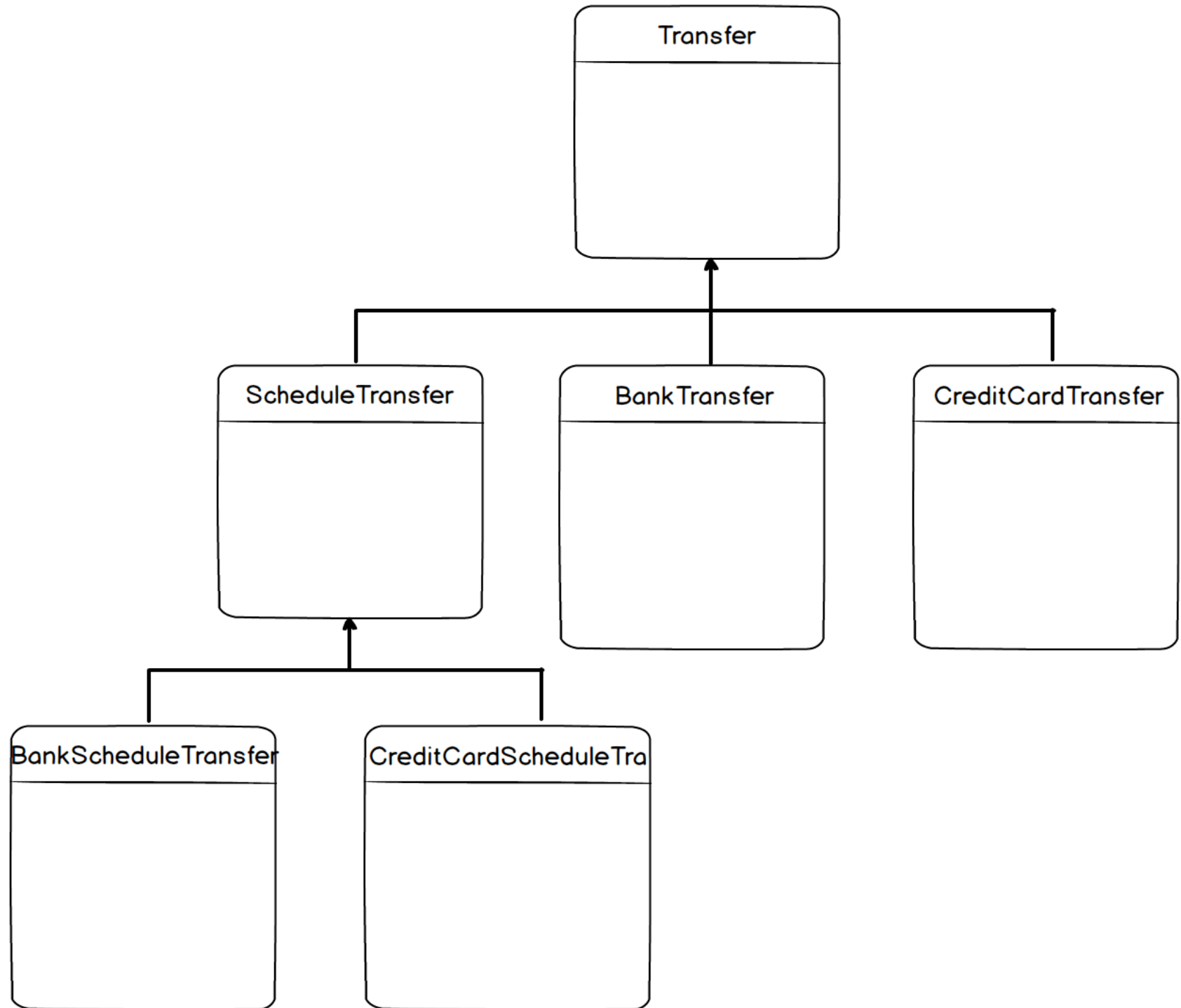
# WZORCE STRUKTURALNE

# Adapter Pattern

Pozwala na współdziałanie ze sobą  
obiektów o niekompatybilnych interfejsach

## Object Adapter







FEATURE

Transfer

ScheduleTransfer

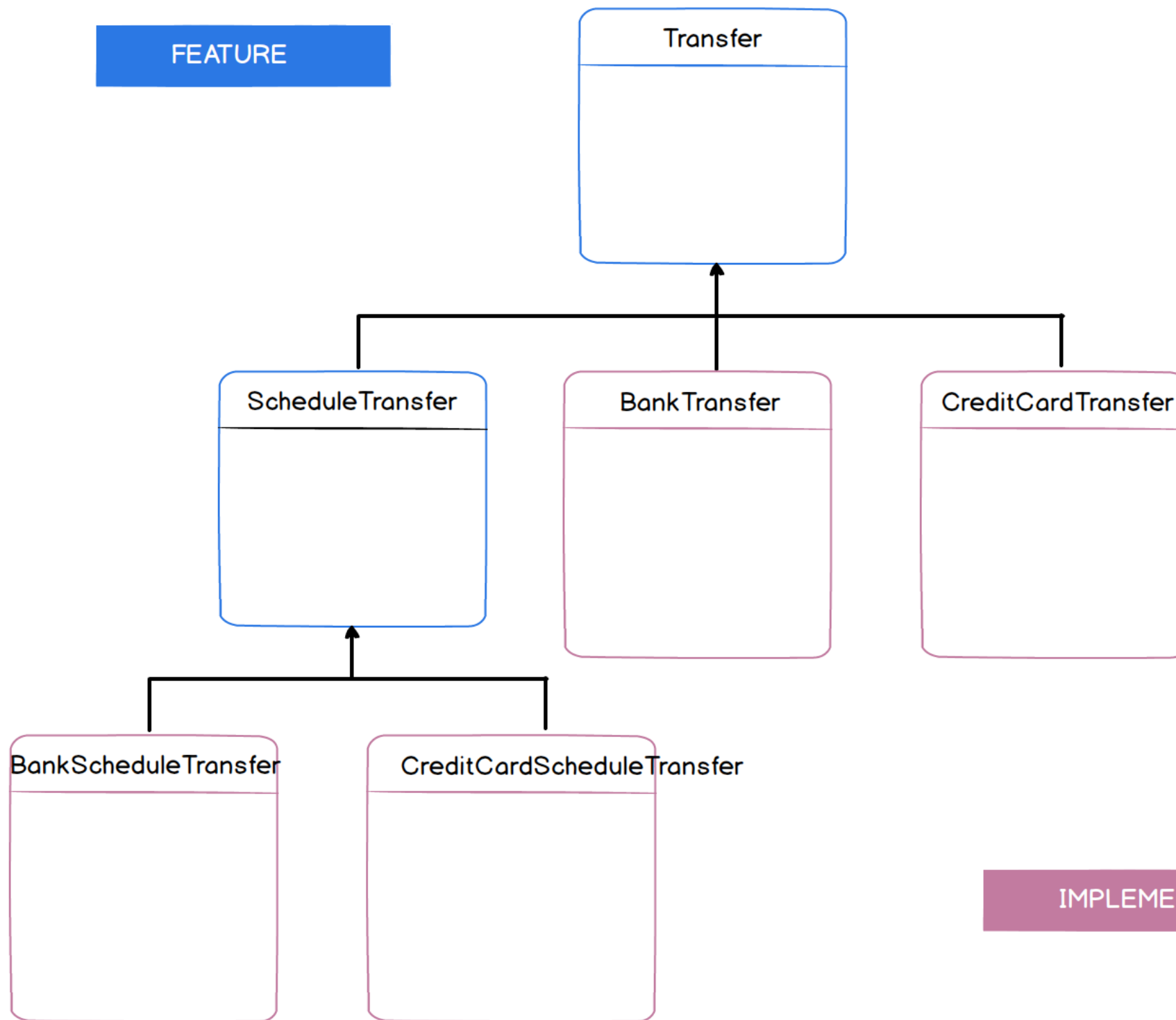
BankTransfer

CreditCardTransfer

BankScheduleTransfer

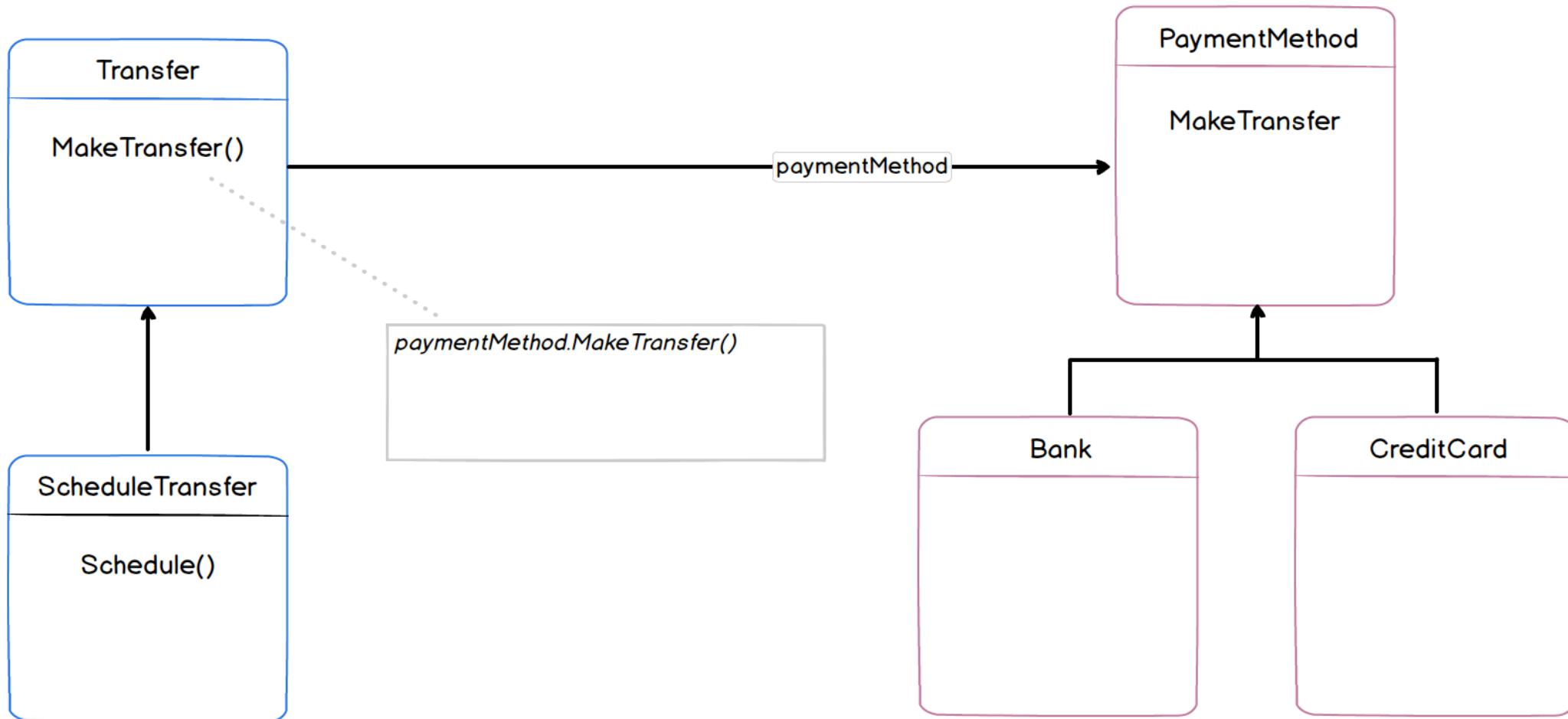
CreditCardScheduleTransfer

IMPLEMENTATION



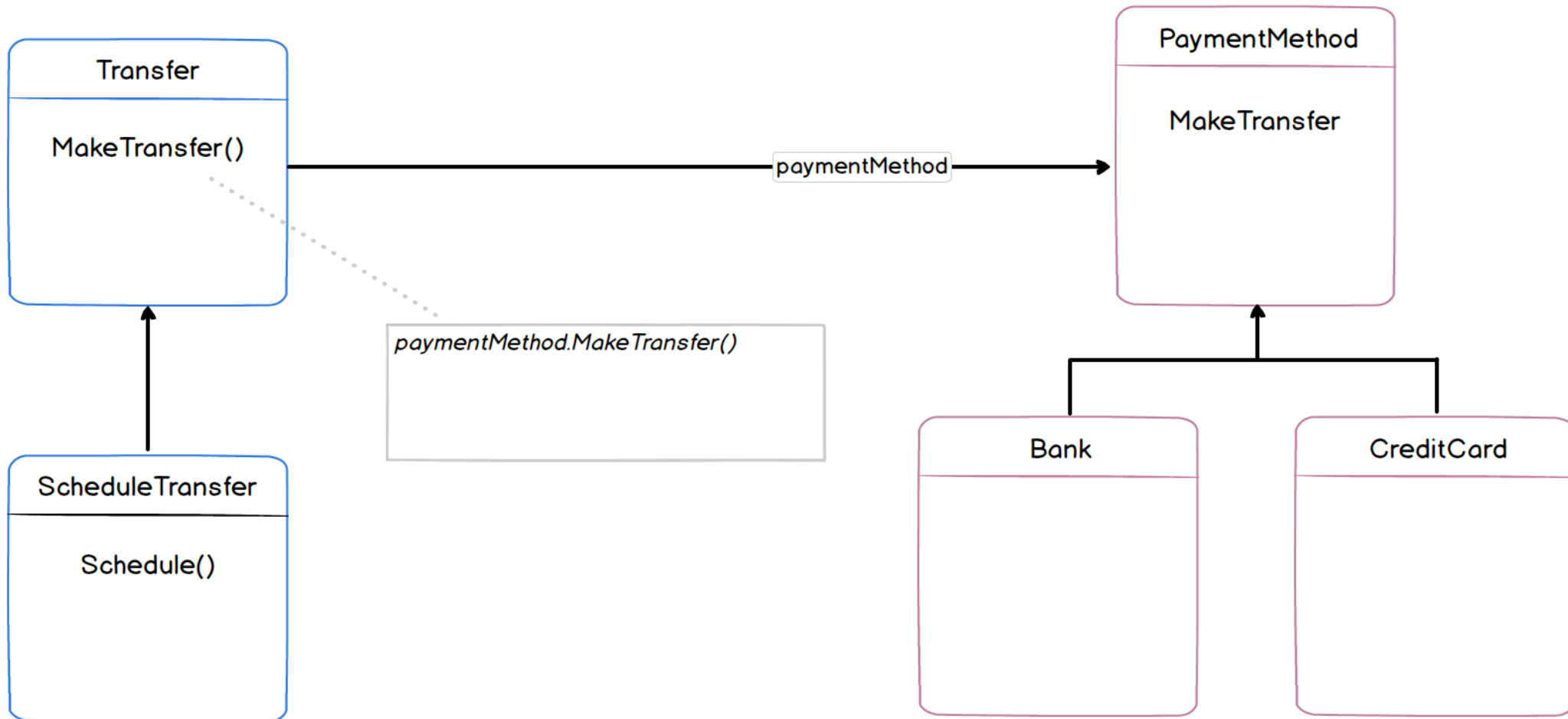
## FEATURE

## IMPLEMENTATION



## FEATURE

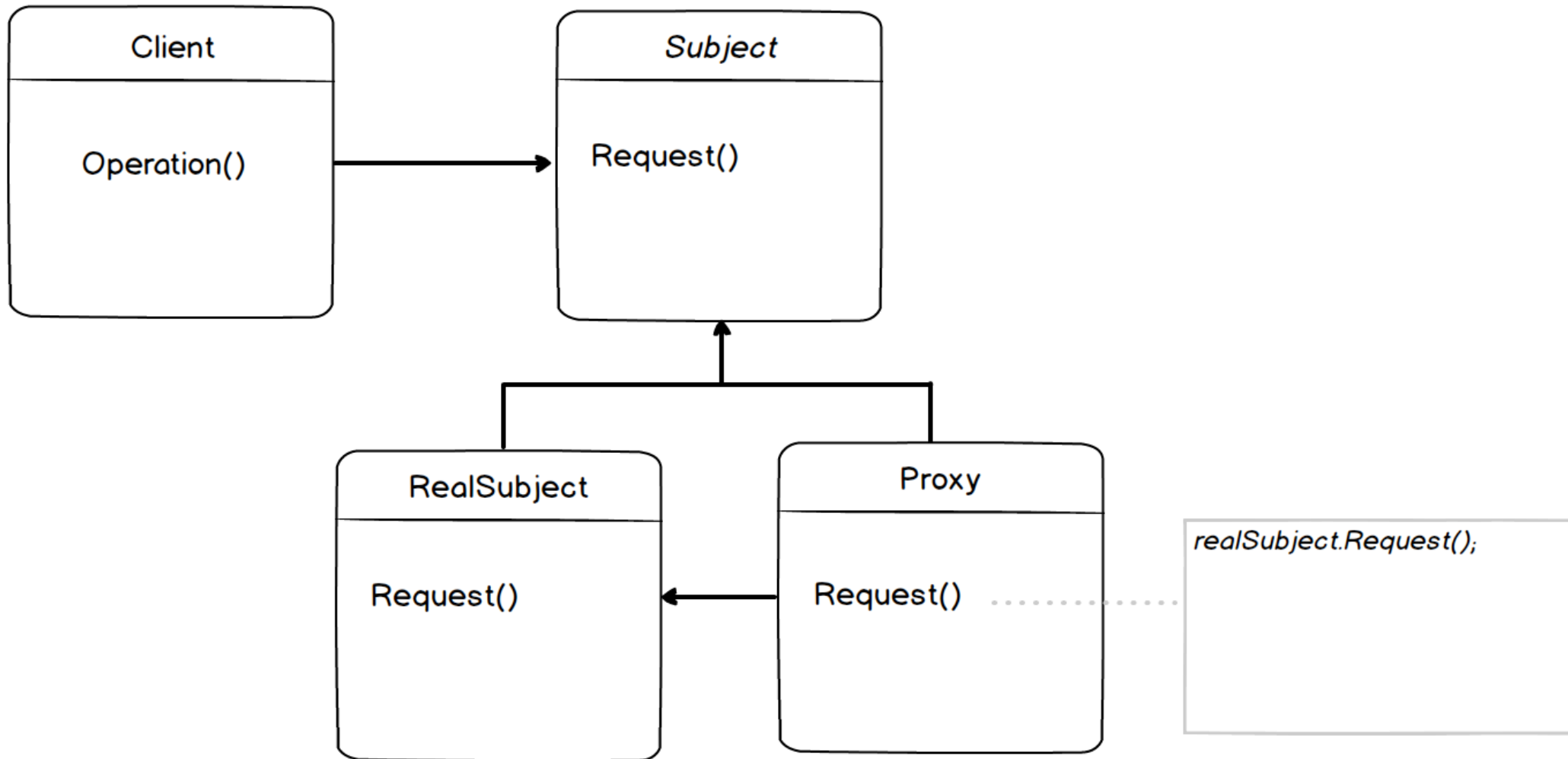
## IMPLEMENTATION



# Proxy Pattern

Pozwala na stworzenie obiektu zastępczego  
w miejsce innego obiektu.

## INTERFACE



# Decorator Pattern

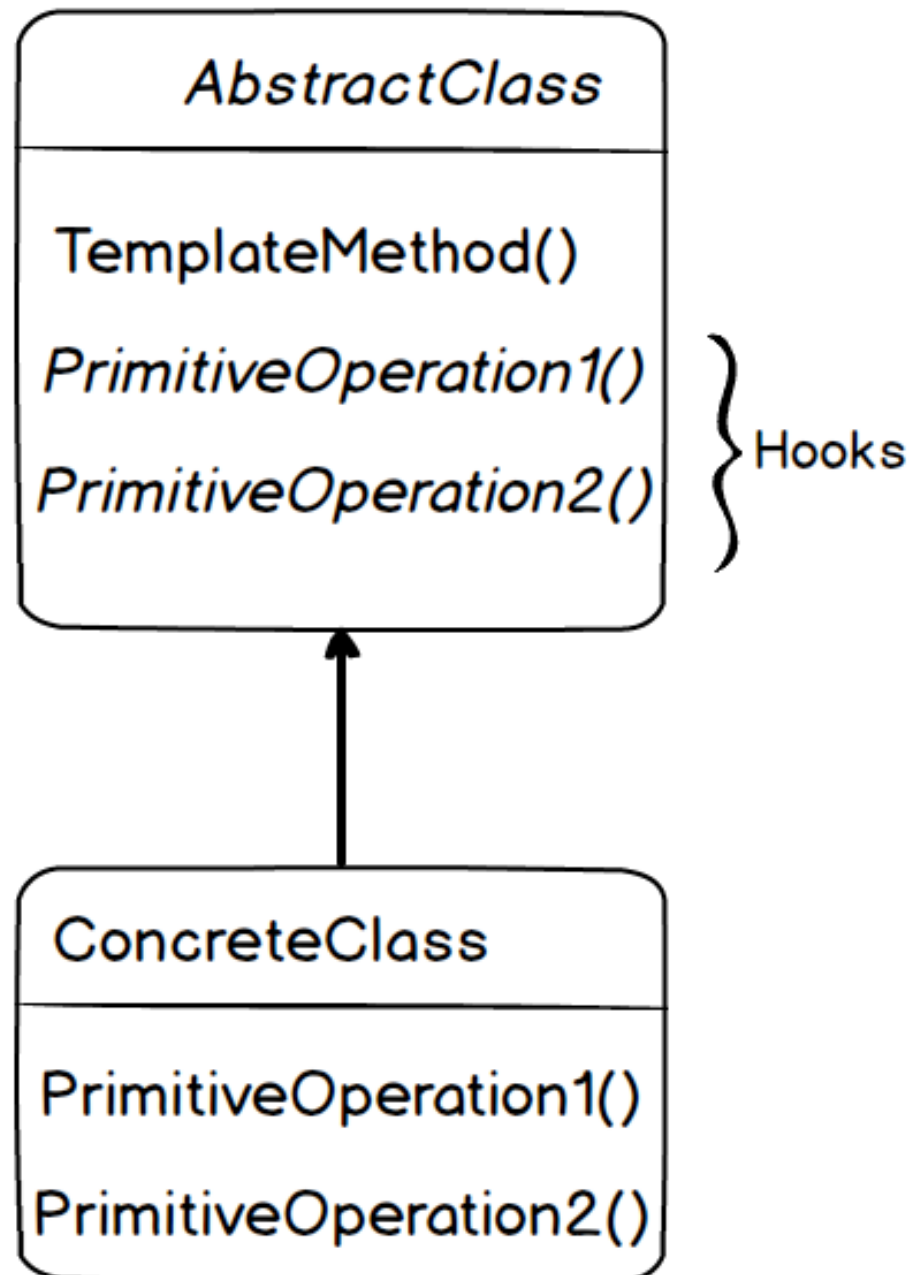
Przedkładaj kompozycję nad dziedziczenie

# WZORCE CZYNNOŚCIOWE

# Template Method

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





# Strategy Pattern

Definiuje rodzinę wymiennych  
algorytmów.

# Fasada Pattern

Ukrywa złożony zestaw klas  
w uproszczony interfejs.

# Command Pattern

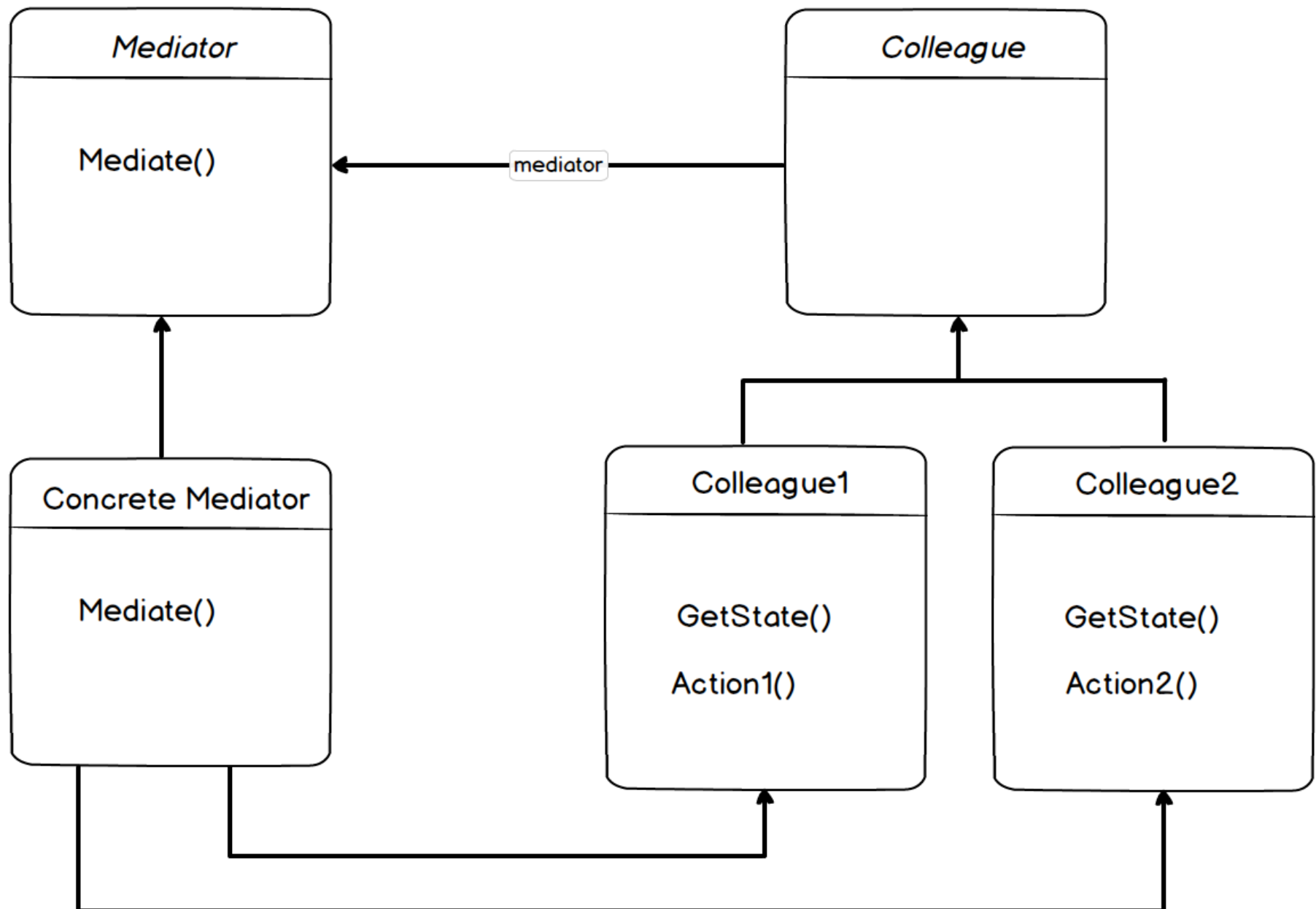
Traktujący żądanie wykonania  
czynności jako **obiekt**.

# Chain of Responsibility Pattern

Pozwala przekazywać żądania wzdłuż  
łańcucha obiektów obsługujących

# Mediator Pattern

Zapewnia jednolity interfejs do różnych elementów danego podsystemu



# Interpreter Pattern

...



# Composite Pattern

Komponuje obiekty w  
struktury drzewiaste

# Iterator Pattern

...

# State Pattern

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

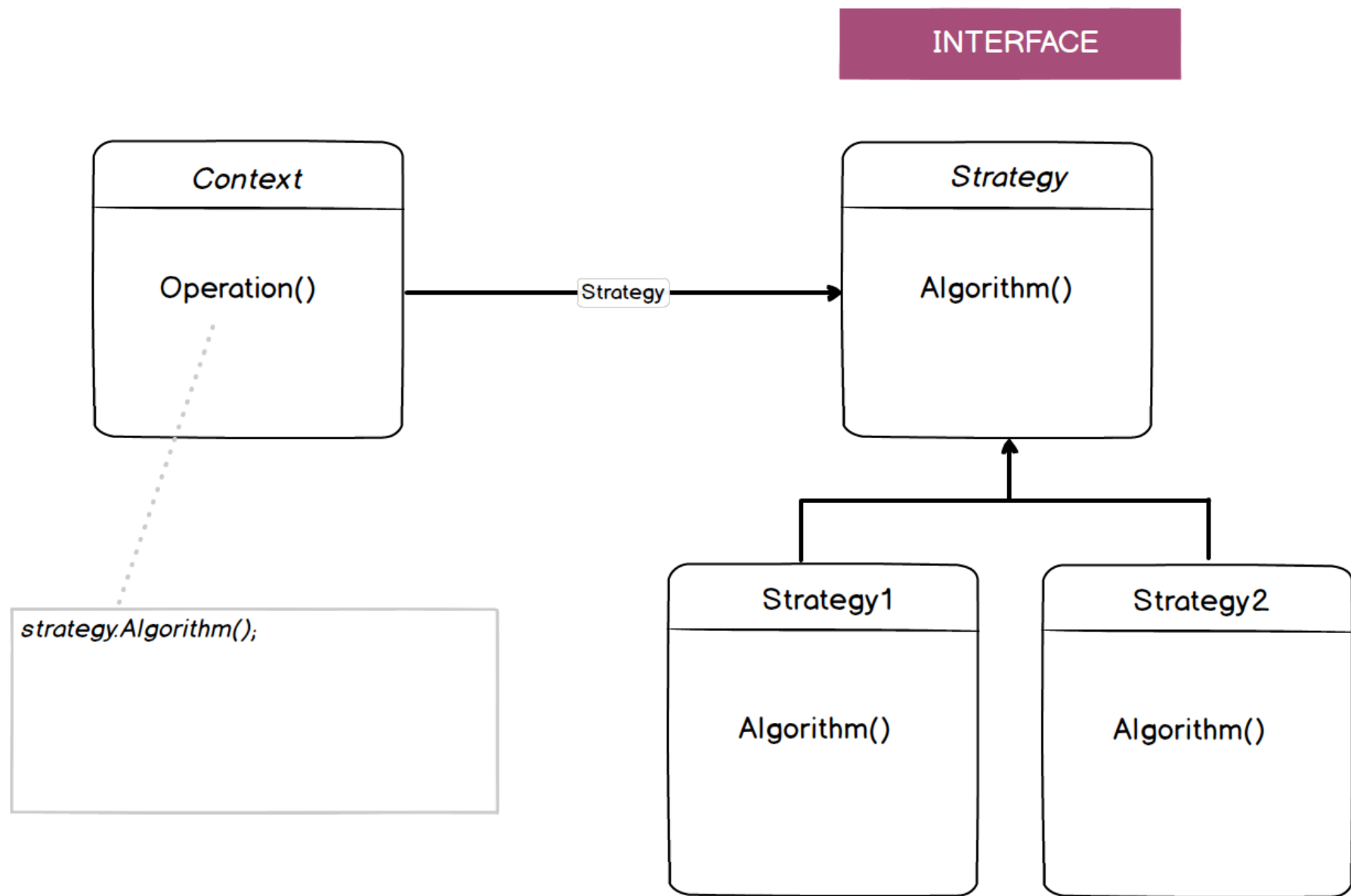
# Memento Pattern

Umożliwia zrobienie migawki stanu obiektu i powrotu do tego stanu.

\_\_\_\_\_

# Null Object Pattern

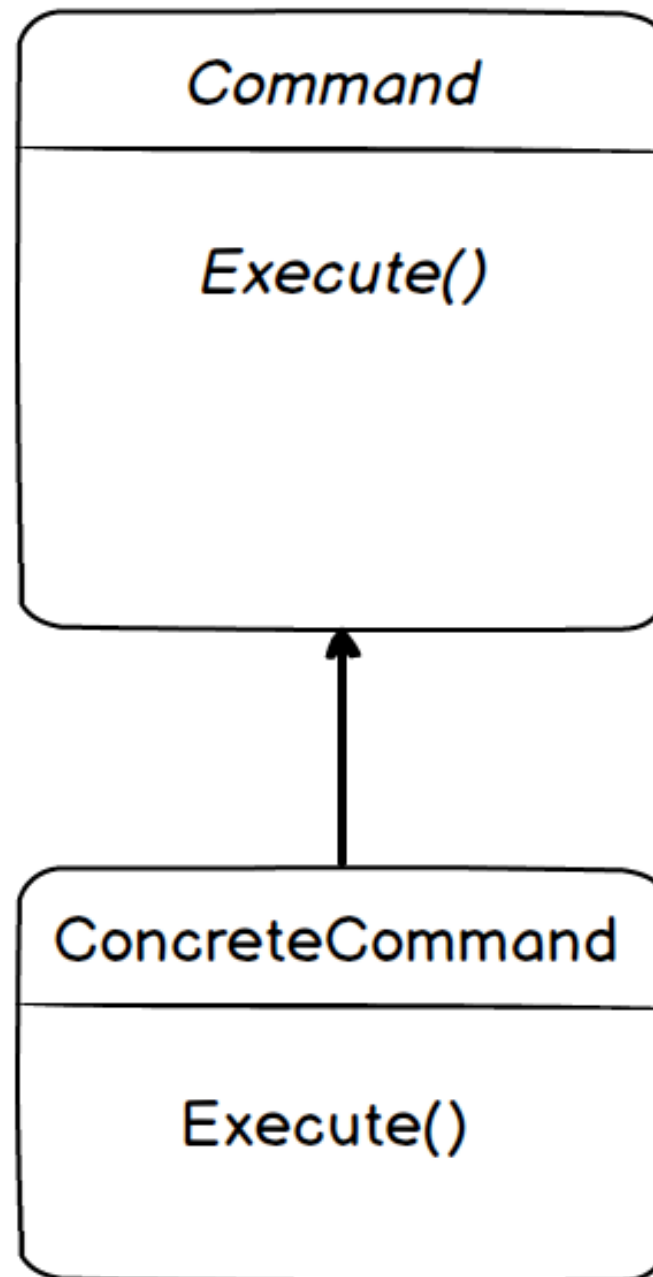
...



# Observer Pattern

**Powiadamiania** zainteresowane obiekty  
o zmianie stanu innego obiektu





# Visitor Pattern

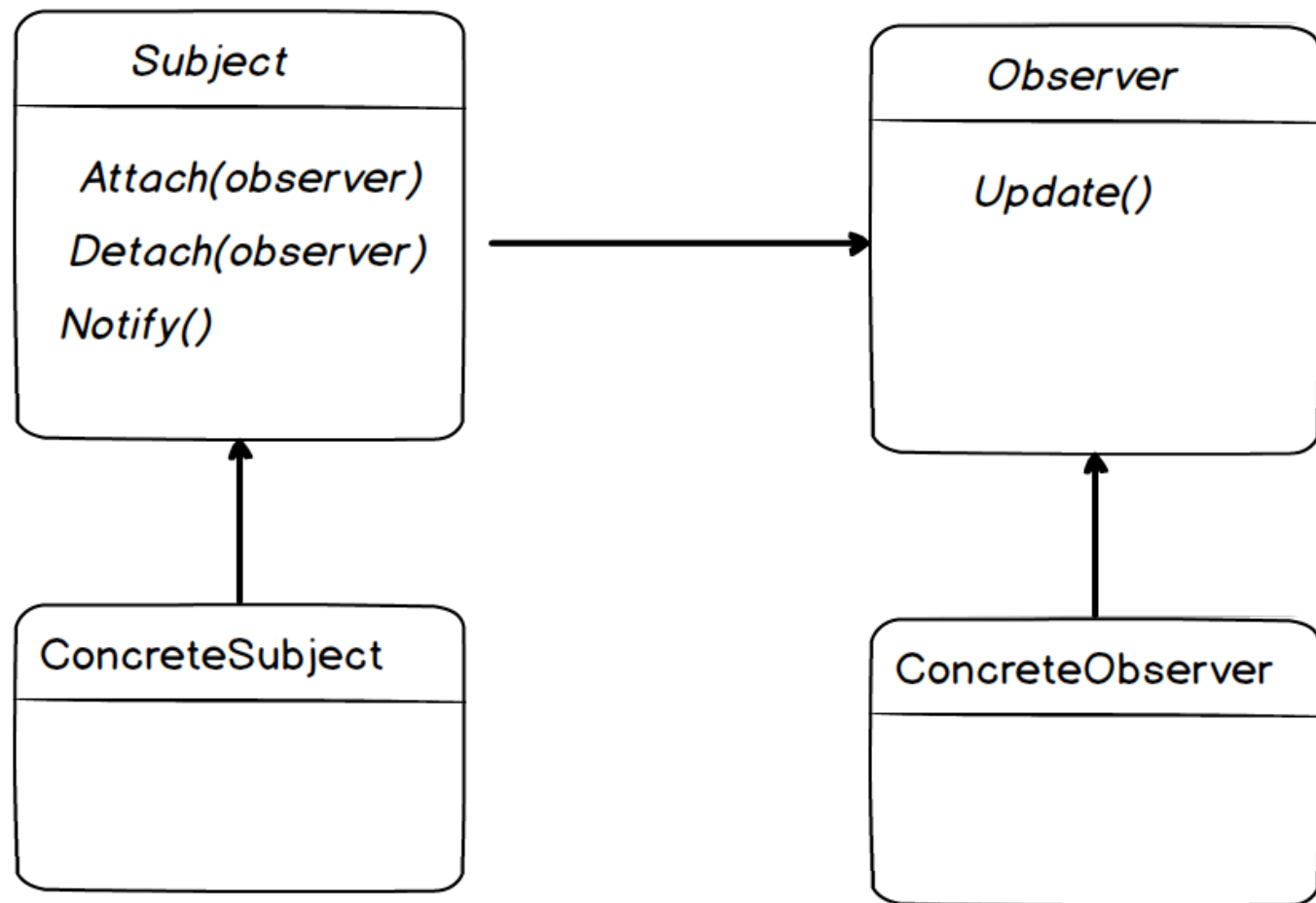
...

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_



\_\_\_\_\_



# Flyweight Pattern

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

