



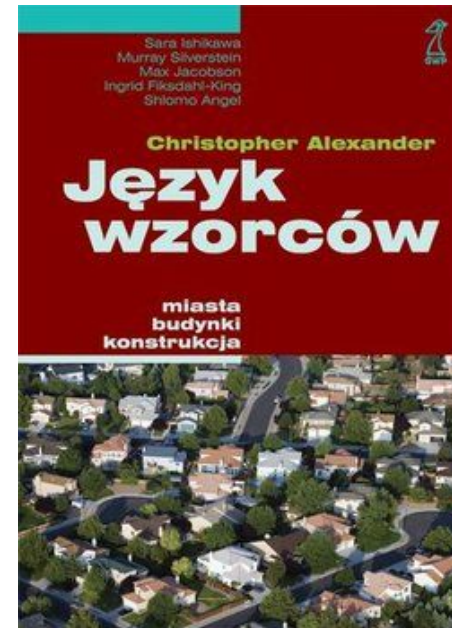
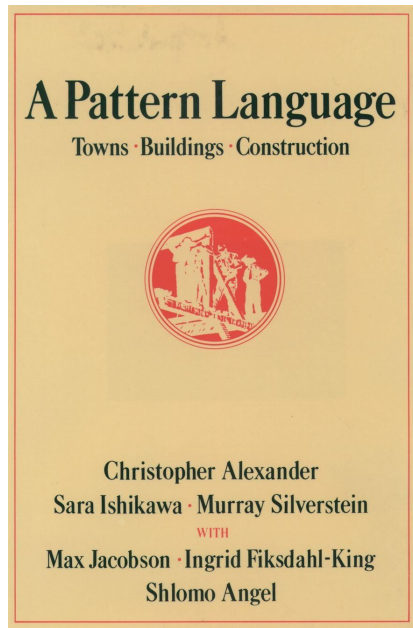
Technologie obiektowe

Wzorce projektowe

Wstęp do wzorców projektowych

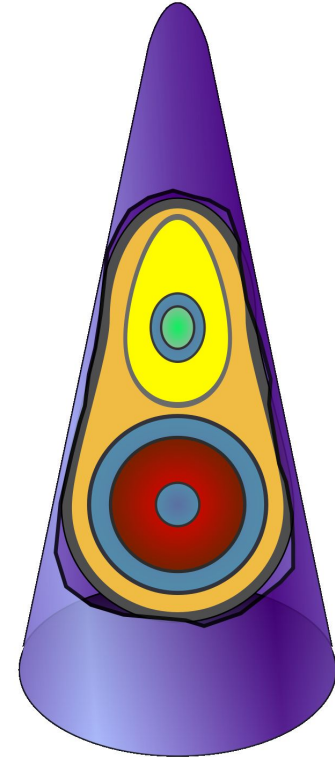
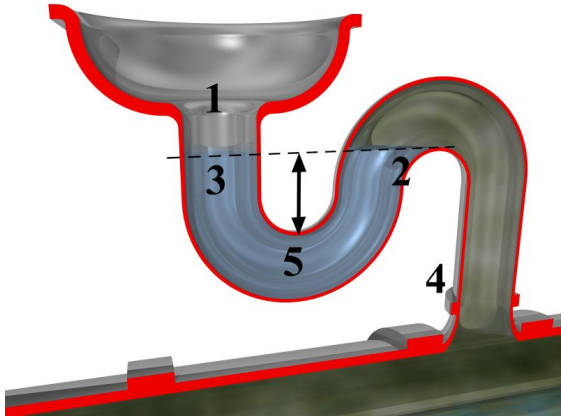


Geneza



Geneza

Wzorzec projektowy - sprawdzone rozwiązanie powszechnego problemu





Wzorce projektowe w inżynierii oprogramowania

Wzorzec projektowy - uniwersalne rozwiązanie często występującego problemu

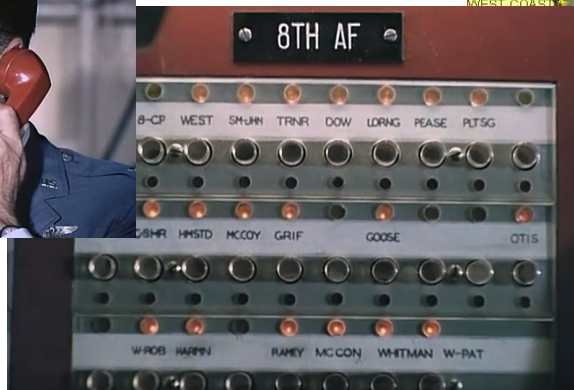
Opis lub szablon rozwiązania problemu

Nie jest samowystarczającym rozwiązaniem

Wyjaśnienie na przykładzie

Odwzorowanie modelu komunikacji globalnej za pomocą fal radiowych

Rzeczywisty system: US HFGCS



HFGCS Station Locations





Model problemu

Stacje w trybie ciągłym nasłuchują transmisji na zdefiniowanych częstotliwościach

Stacje znają znaczenie poszczególnych komunikatów

W zależności od komunikatu wdrażane jest odpowiednie postępowanie



Model problemu: przykład

ALL STATIONS THIS IS JANTO BREAK L4TGNF YUW23FSGXGST678QWERTYUIOP

THIS IS MAINSAIL STANDING BY FOR TRAFFIC

THIS IS MAINSAIL WITH A TEST COUNT TESTING 12345 54321

ALL STATIONS THIS IS JANTO FOR BALLSACK BREAK MB6DL4 YUSFGA6789QWERXCVSER
DISREGARD THIS TRANSMISSION



Model: rozwiązanie

Podział na dwie grupy abonentów:

Obserwator

Obserwujący

Wzorzec projektowy: Obserwator

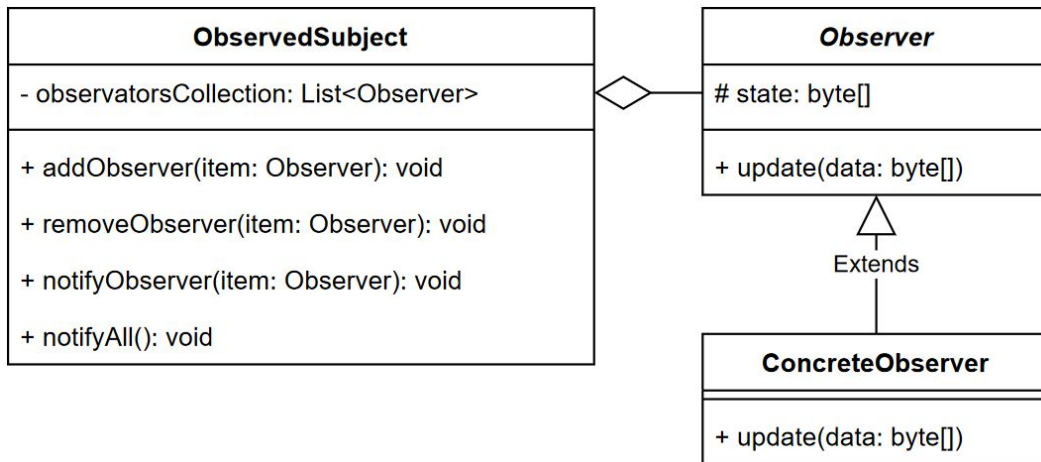
Obserwator: observer

Podmiot: subject

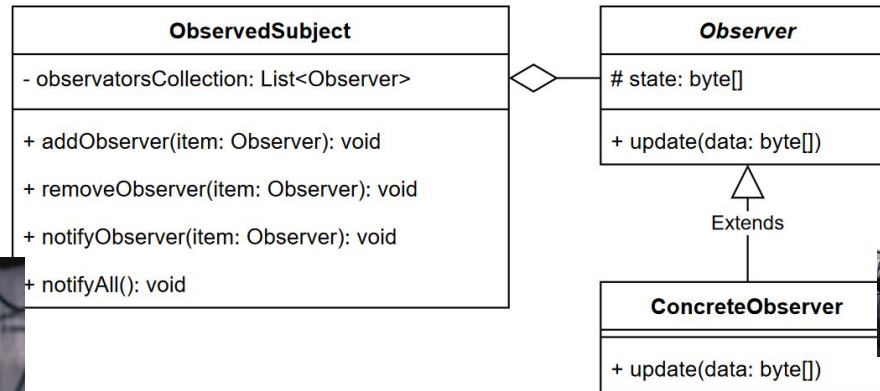


Obserwator

W momencie zmiany stanu podmiotu, wszyscy podlegający zostają poinformowani



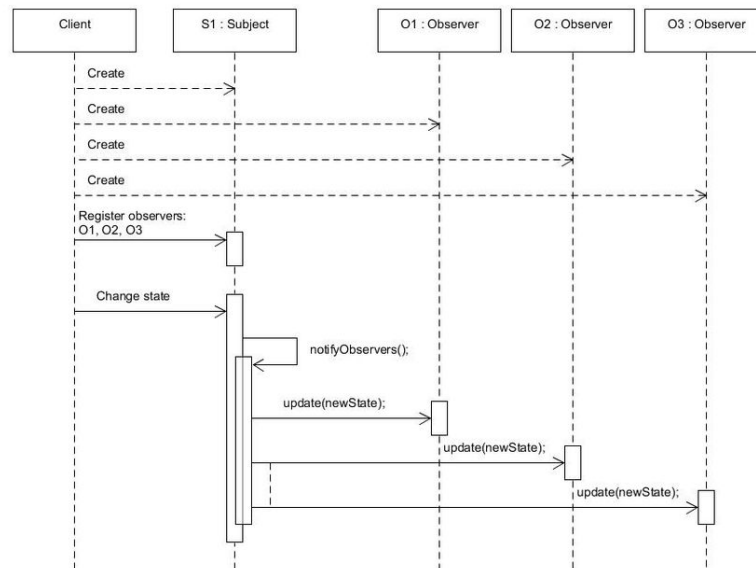
Observer



Obserwator

Rejestracja obserwatorów O1,O2, O3

W momencie zmiany stanu podmiotu, obserwatorzy zostają powiadomieni





Ponowne wykorzystanie kodu

Powiadamianie wątków

Model selektywnego wywołania PSP

Rendering kontrolek przy zmianie rozmiaru

Rendering przy widoku danych przy zmianie danych



Ponowne wykorzystanie

System: wspólne przetwarzanie dokumentów

Dokumenty Google,

Arkusze Google, etc.



Jak używać wzorców projektowych

Poznać wzorce

Poznać problemy rozwiązywane przez te wzorce

Rozpoznać kiedy problem jest rozwiązywalny przez wzorzec

Dostosować wzorzec do problemu

Łączyć wzorce



Wzorzec projektowy

Nazwa wzorca projektowego

Rozwiązany problem

Uczestnicy

Struktura:

- organizacja kla

- współdziałanie klas



Wzorce projektowe

Kreacyjne

Behawioralne

Strukturalne



Wzorce kreacyjne

Mechanizmy tworzenia obiektów - tworzenie właściwe w danej sytuacji

Właściwe przy zmianach w modelowanym systemie

Singleton

Metoda fabrykująca

Fabryka abstrakcyjna

Budowniczy

Prototyp



Wzorce strukturalne

Ułatwiają projektowanie przez identyfikację prostej drogi określania relacji między encjami

Komponują struktury z mniejszych obiektów

Adapter

Dekorator

Most

Kompozyt

Fasada

Pyłek

Pośrednik



Wzorce behawioralne

Bazują na komunikacji

Odwzorowują skomplikowane przepływy sterowania

Łańcuch zobowiązań

Iterator

Polecenie

Mediator

Pamiętka

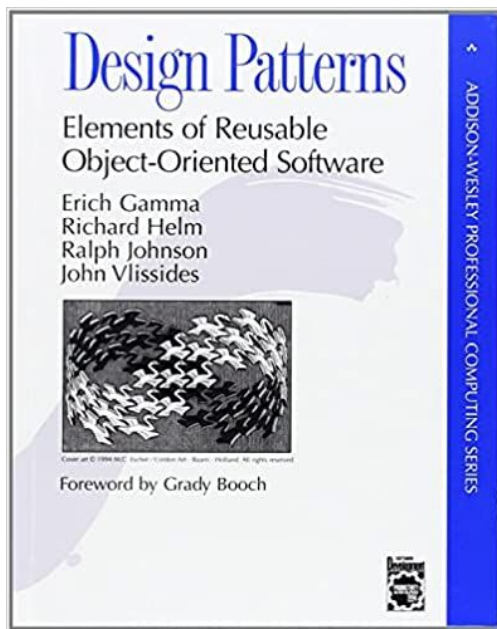
Obserwator

Stan

Strategia

Metoda szablonowa

Gang of four



Fabryka abstrakcyjna

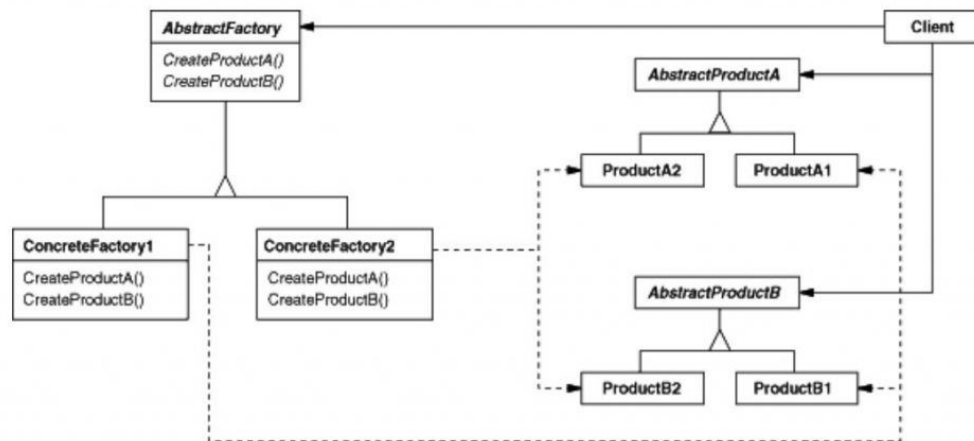
Cel: Zapewnienie interfejsu do tworzenia rodzin powiązanych lub zależnych obiektów bez określenia ich konkretnych klas.

Zastosowanie:
system niezależny od procesu tworzenia produktów
ich kompozycji oraz reprezentacji

system konfigurowany z co najmniej
jedną rodziną produktów

rodzina obiektów powiązanych z sobą
produktów jest przeznaczona do użytku razem
i istnieje konieczność wymuszenia tego faktu

dostarczenie biblioteki produktów
przy ujawnieniu jedynie interfejsu



Fabryka abstrakcyjna

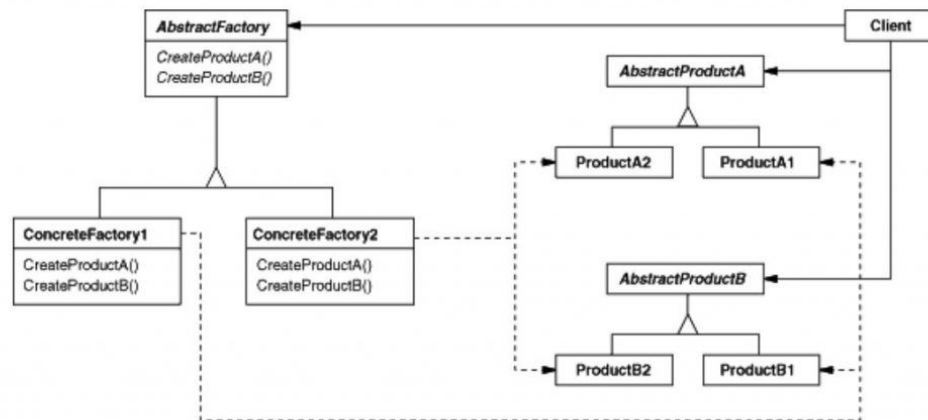
Abstract factory:
deklaruje interfejs dla operacji tworzących obiekty abstrakcyjnego produktu.

ConcreteFactory:
realizuje operacje tworzenia obiektów konkretnych produktów.

Abstract product:
deklaruje interfejs dla typu obiektu produktu.

ConcreteProduct:
definiuje obiekt produktu, który ma zostać stworzony przez odpowiedni implementuje interfejs AbstractProduct.

Client:
używa tylko interfejsów zadeklarowanych przez AbstractFactory i AbstractProduct





Fabryka abstrakcyjna

Izoluje interfejs od implementacji.

Ułatwia wymiennność rodzin produktów.

Promuje spójność między produktami.

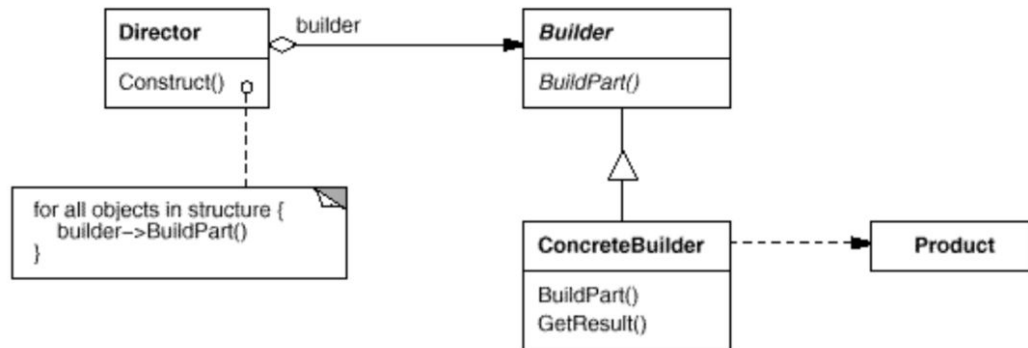
Utrudnia rozbudowę.

Budowniczy

Cel: Oddzielenie procesu tworzenia złożonego obiektu od jego reprezentacji, w celu budowy różnych reprezentacji przez jeden proces.

Zastosowanie:
algorytm tworzenia obiektu złożonego powinien być niezależny od części, które składają się na obiekt i sposobu ich złożenia.

proces musi umożliwiać budowę różnych reprezentacji obiektu.



Budowniczy

Builder:
określa abstrakcyjny interfejs do tworzenia części obiektu Product.

ConcreteBuilder:
konstruuje i składa części produktu poprzez implementację interfejsu Builder.

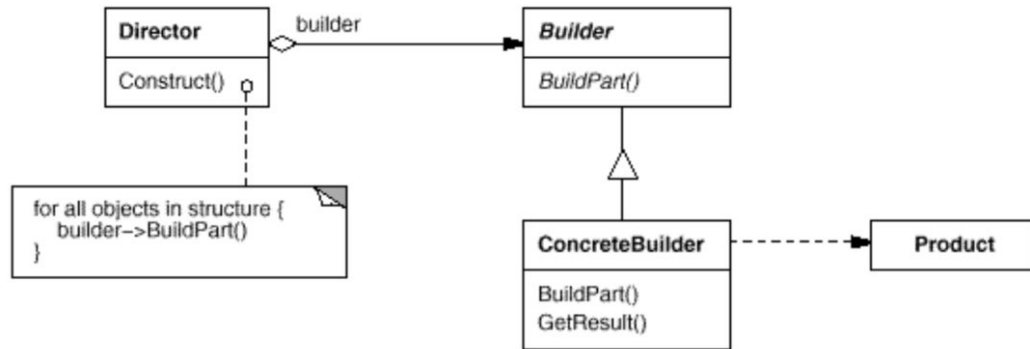
definiuje i śledzi reprezentację, którą tworzy.

zapewnia interfejs do pobierania produktu.

Director:
buduje obiekt za pomocą Builder

Product:

Reprezentuje złożony produkt i jego części

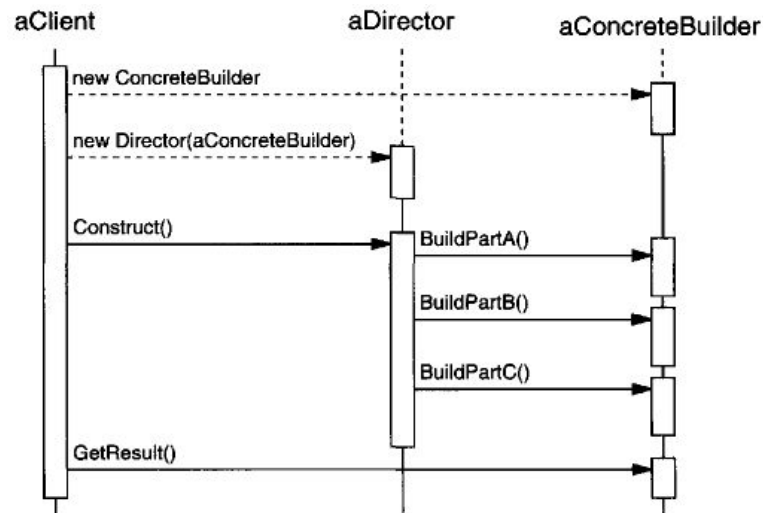


Budowniczy

Umożliwia zróżnicowanie reprezentacji.

Izoluje kod od reprezentacji.

Sprawia kontrolę nad procesem bardziej dokładną.



Metoda wytwórcza

Cel:

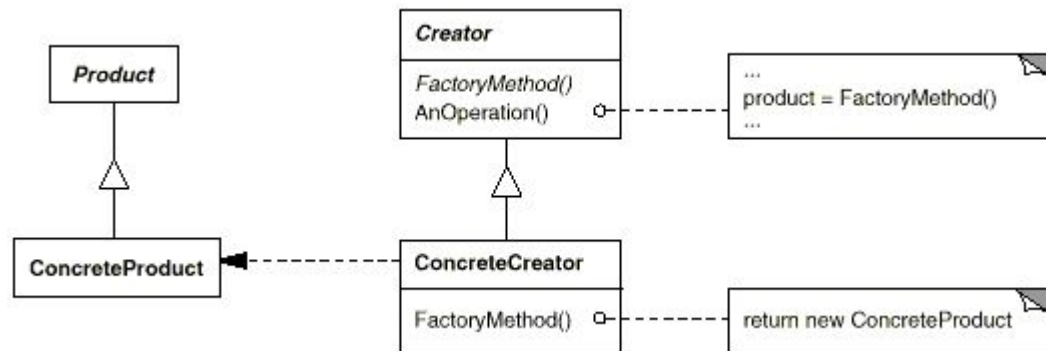
Definicja interfejsu tworzenia obiektu ale to podklasy decydują, którą klasę zainstancjonować. Odracza tworzenie instancji.

Zastosowanie:

klasa nie ma możliwości przewidywania jakie klasy ma instancjonować.

klasa oczekuje podklas do określenia obiektu, który tworzy.

klasy delegują odpowiedzialności do podklas



Metoda wytwórcza

Product:
definiuje interfejs obiektów, które tworzy metoda wytwórcza.

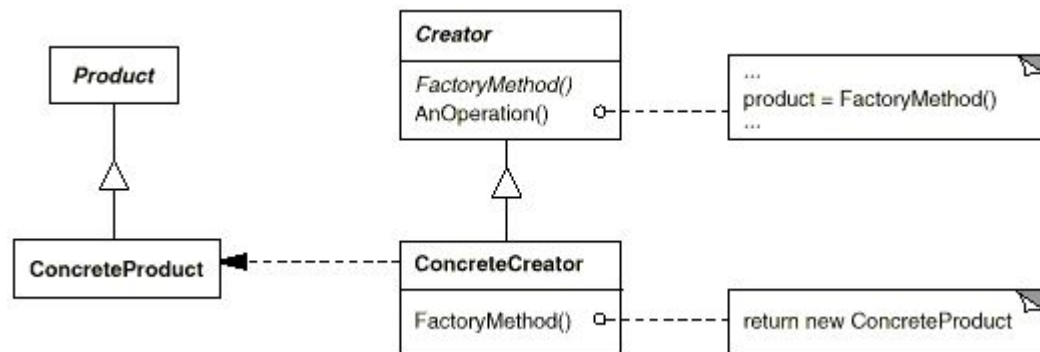
ConcreteProduct:
implementuje interfejs produktu.

Creator:
deklaruje metodę fabryki, która zwraca obiekt typu Product.

może również zdefiniować domyślną implementację metody wytwórczej, która zwraca domyślny obiekt ConcreteProduct.

może wywołać metodę fabryki w celu utworzenia obiektu Product.

ConcreteCreator:
nadpisuje metodę fabryki, aby zwrócić instancję ConcreteProduct



Prototyp:

Cel:

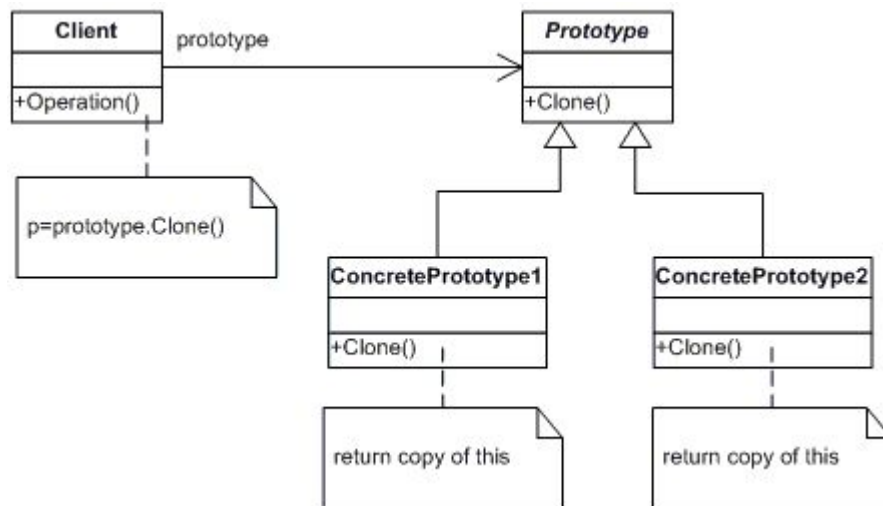
Umożliwienie kopiowania obiektów przy użyciu prototypowej instancji.

Zastosowanie:

w sytuacji, w której system jest niezależny od sposobu tworzenia, składania i reprezentacji

gdy klasy przewidziane do instancjonowania są znane na etapie wykonania

gdy instancje mogą mieć tylko kilka stanów

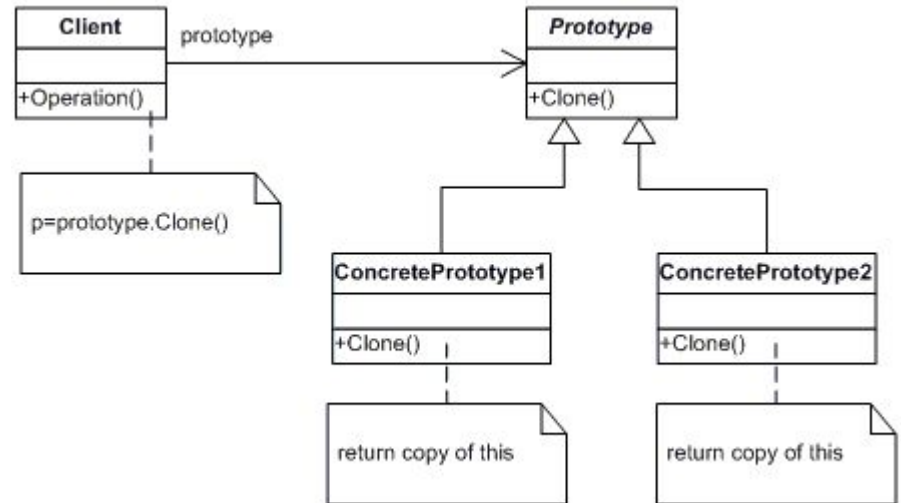


Prototyp:

Prototype:
deklaruje interfejs klonowania

ConcretePrototypeN:
implementuje operację klonowania

Client:
żąda sporządzenia kopii



Adapter

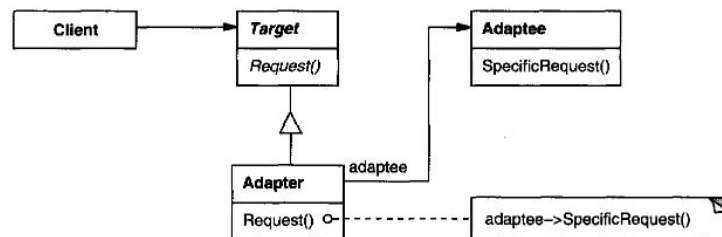
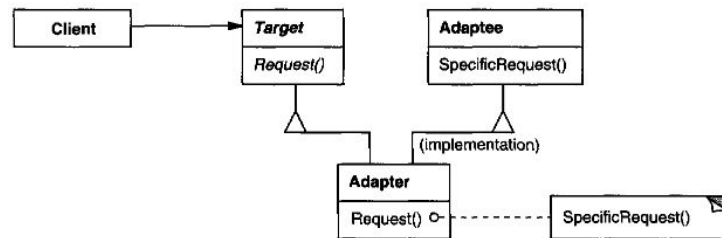
Cel:

Zmiana interfejsu na ten żądany przez użytkownika.

Zastosowanie:

Chęć wykorzystania istniejących klas z niekompatybilnym interfejsem.

Chęć zaprojektowania klasy wielokrotnie wykorzystywanej przy nieznanym interfejsie przyszłych interfejsów



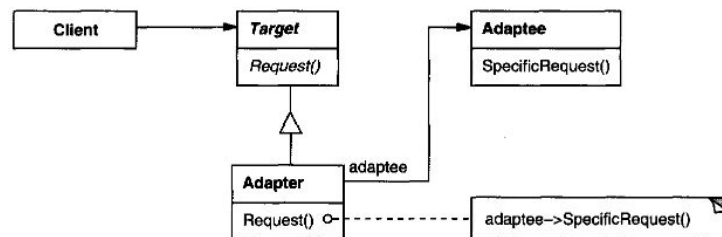
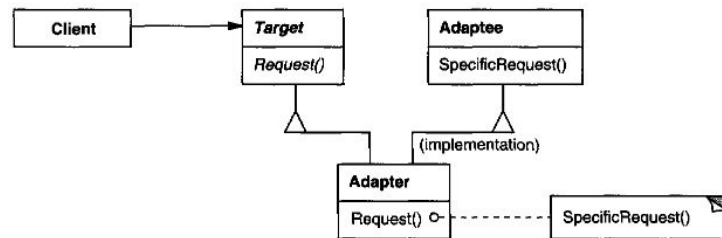
Adapter

Target:
Interfejs zależny od dziedziny.

Adapter:
Dostosowuje interfejs do wymagań.

Adpatee:
Implementuje istniejący interfejs zależny od dziedziny

Client:
Wykorzystuje interfejs



Kompozyt

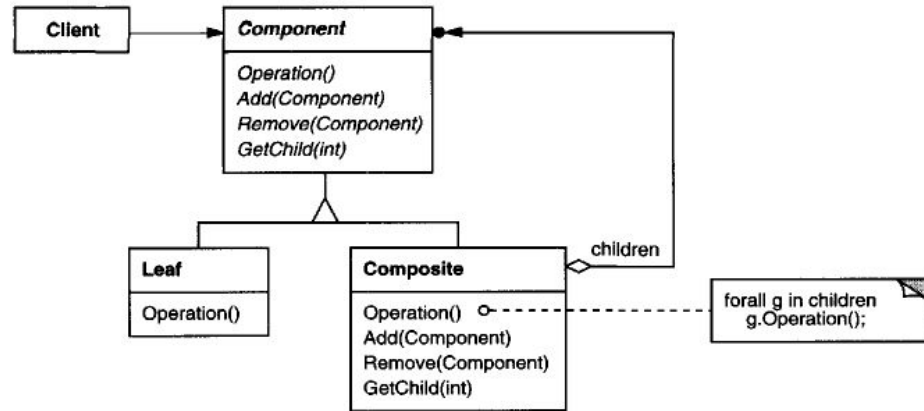
Cel:

Odwzorowanie struktury drzewiastej.

Relacja całość-część.

Zastosowanie:

Zignorowanie różnic między całością i częściami.



Kompozyt

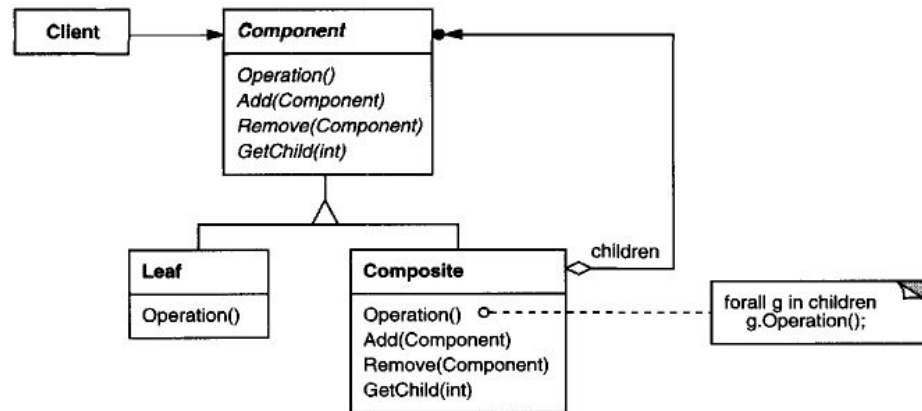
Component
deklaruje interfejs obiektów zbioru
implementuje domyślne zachowanie
deklaruje interfejs manipulacji węzłami

Leaf
reprezentuje liść

definiuje zachowanie elementów prostych

Composite
definiuje zachowanie elementów posiadających liście

Client
manipuluje elementami

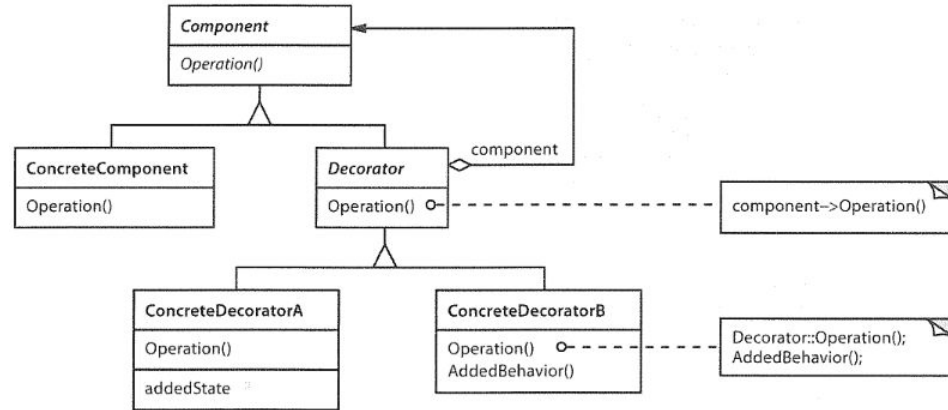


Dekorator

Cel: dynamiczne dołączenie odpowiedzialności do obiektu.

Zastosowanie:
konieczność dołączenia odpowiedzialności do obiektu w transparentny i dynamiczny sposób

niepraktyczność dziedziczenia



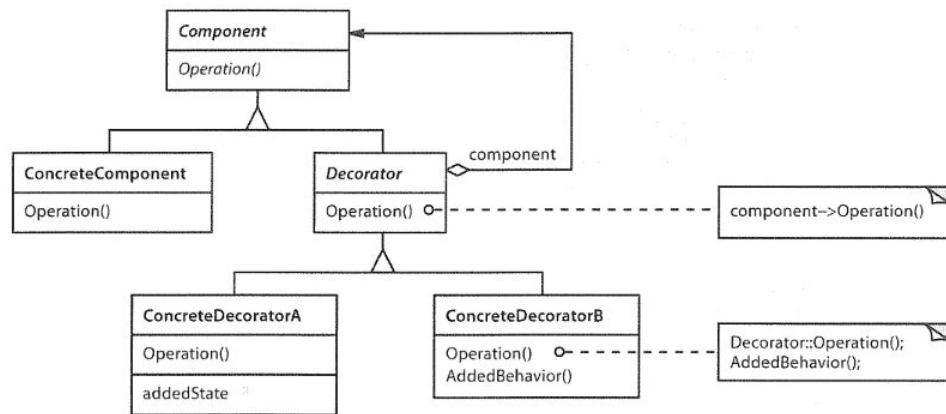
Dekorator

Component
definicja interfejsu obiektów o znanej funkcjonalności

ConcreteComponent
implementacja interfejsu obiektów o znanej funkcjonalności

Decorator
przechowuje referencje do obiektu komponentu (z zachowaniem interfejsu)

ConcreteDecorator
dodaje odpowiedzialność do komponentu



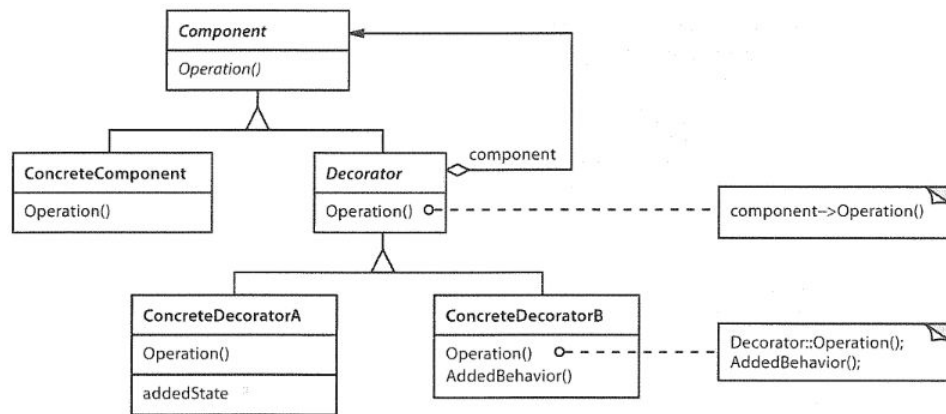
Dekorator

Większa elastyczność niż dziedziczenie

Unikanie budowania skomplikowanych klas

Niewidoczna otoczka

Skomplikowanie



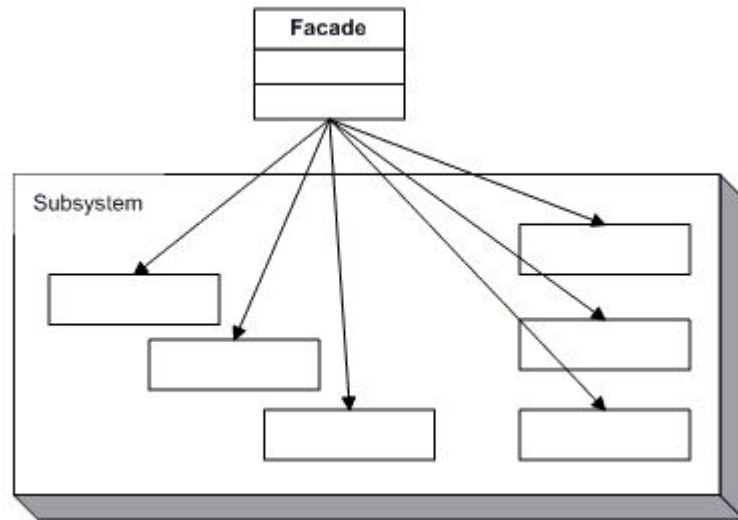
Fasada

Cel: udostępnienie jednolitego interfejsu dla zbioru interfejsów podsystemu

Zastosowanie:
dostarczenie jednolitego interfejsu dla złożonego systemu

duża liczba zależności między klasami

podział na warstwy



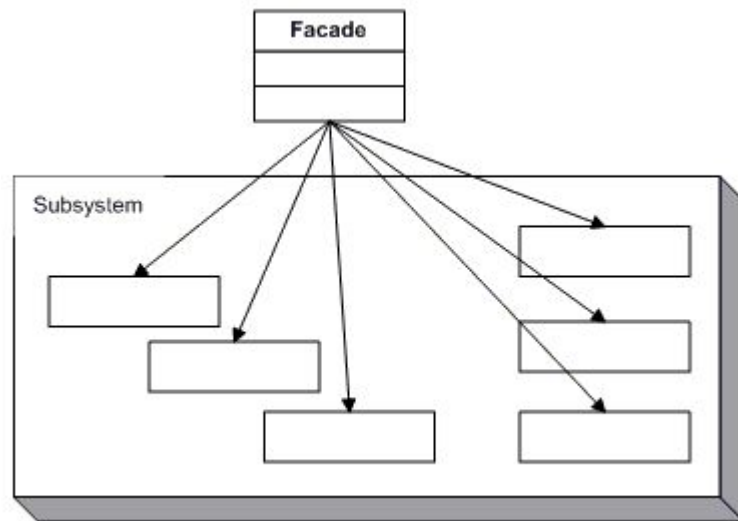
Fasada

Facade

zna interfejsy elementów podsystemu
deleguje operacje do wykonania do ww elementów

Klasy podsystemu

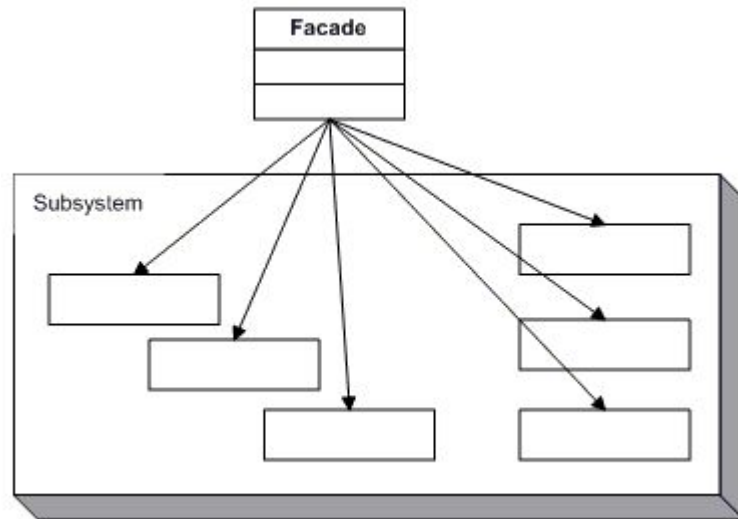
implementują funkcjonalności podsystemu
obsługują żądania fasady
nie wiedzą o istnieniu fasady



Fasada

Oddzielenie klientów od podsystemu

Rozluźnienie powiązania

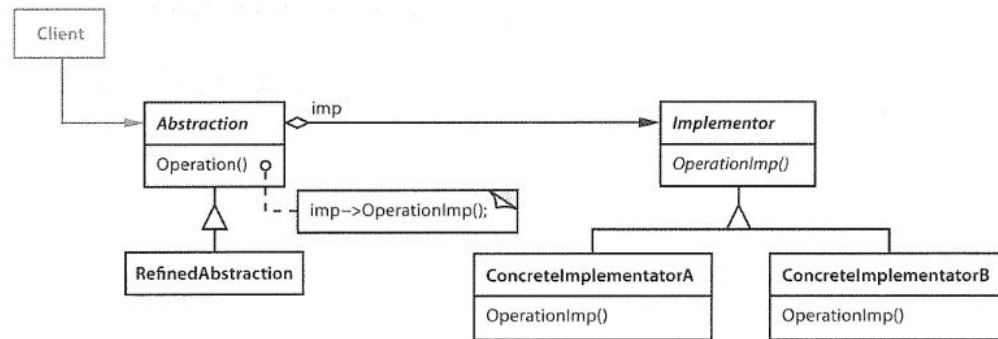


Most

Cel: oddzielenie abstrakcji od implementacji

Zastosowanie:
uniknięcie trwałego powiązania abstrakcji i implementacji

umożliwienie rozszerzenia abstrakcji i implementacji



Most

Abstraction

Definiuje interfejs abstrakcji

RefinedAbstraction

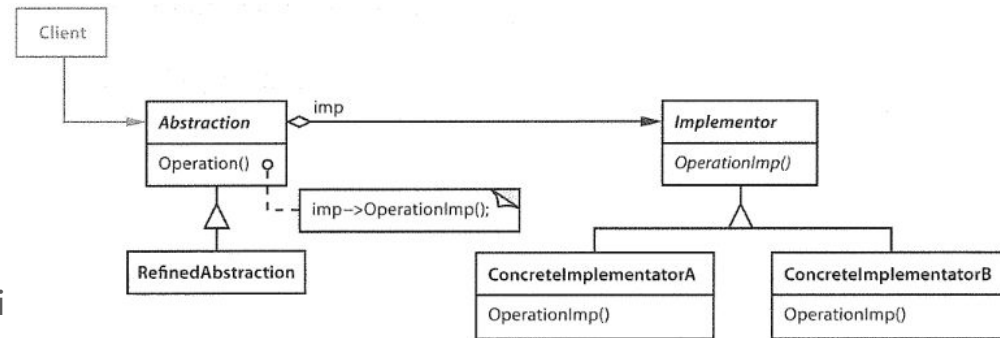
rozszerza interfejs abstrakcji

Implementor

definiuje interfejs klas po stronie implementacji

ConcreteImplementor

konkretna implementacja

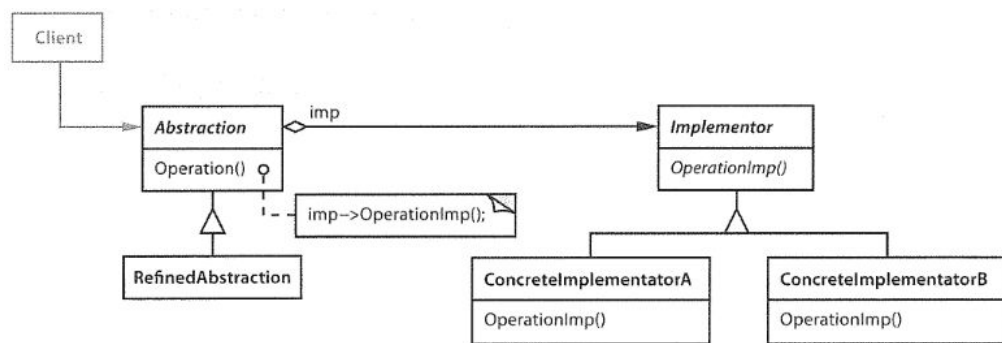


Most

Oddzielenie interfejsu od implementacji

Ułatwione rozszerzanie

Ukrywanie implementacji



Pełnomocnik/Pośrednik

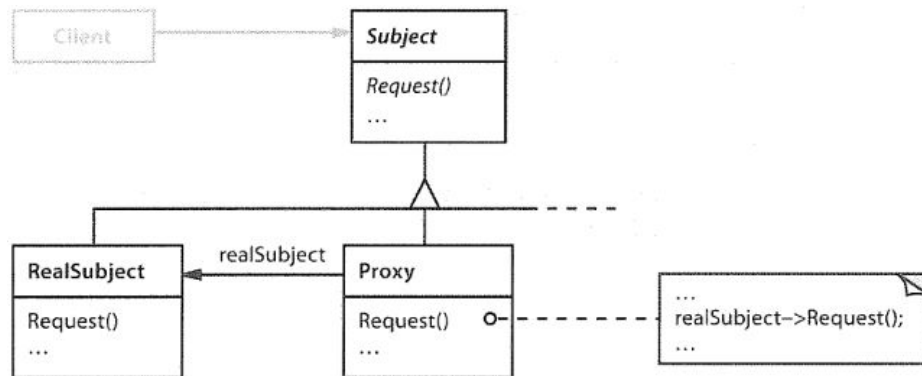
Cel: pośrednictwo w dostępie

Zastosowanie:
lokalna reprezentacja obiektu z innej przestrzeni adresowej

tworzenie na żądanie skomplikowanych obiektów

kontrola dostępu do obiektu pierwotnego

implementacja inteligentnej referencji



Pełnomocnik/Pośrednik

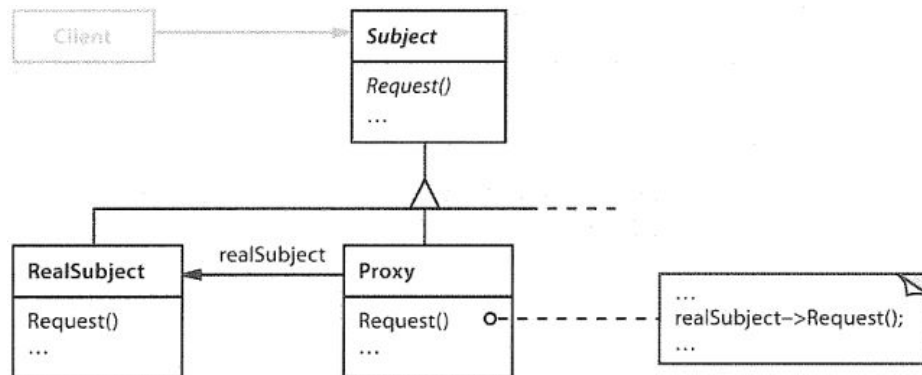
Proxy:

przechowuje referencję rzeczywistego obiektu
udostępnia interfejs identyczny z Subject
kontroluje dostęp do rzeczywistego obiektu lub
koduje i przekazuje żądania i argumenty do

rzeczywistego obiektu w innej przestrzeni
adresowej

przechowywanie informacji dodatkowych
o obiekcie

sprawdzenie uprawnień dostępu



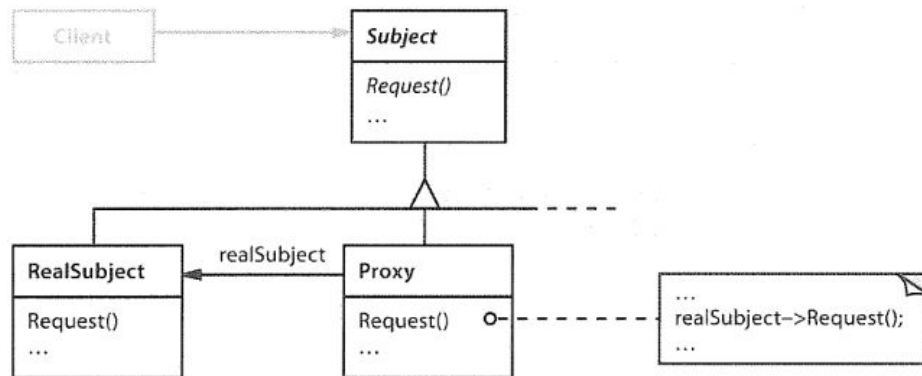
Pełnomocnik/Pośrednik

Subject:

definiuje wspólny interfejs (RS i P)

RealSubject:

reprezentuje obiekt rzeczywisty

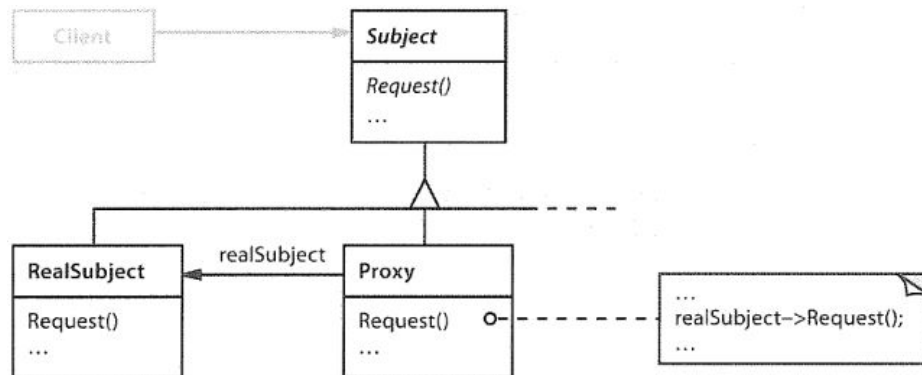


Pełnomocnik/Pośrednik

Ukrycie faktu rzeczywistej przestrzeni adresowej

Możliwa optymalizacja

Dodatkowe operacje przed delegacją



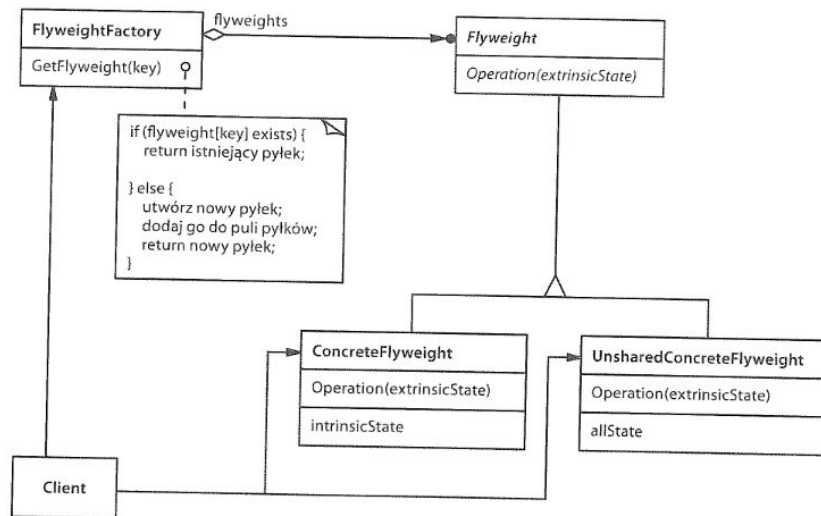
Pyłek

Cel: współdzielenie celem wydajnej obsługi
dużej liczby małych obiektów

Zastosowanie:
wykorzystanie dużej liczby małych obiektów

koszt przechowywania obiektów jest duży ze
względu na liczbę instancji

większość stanu obiektu przechowywana poza
nimi



Pyłek

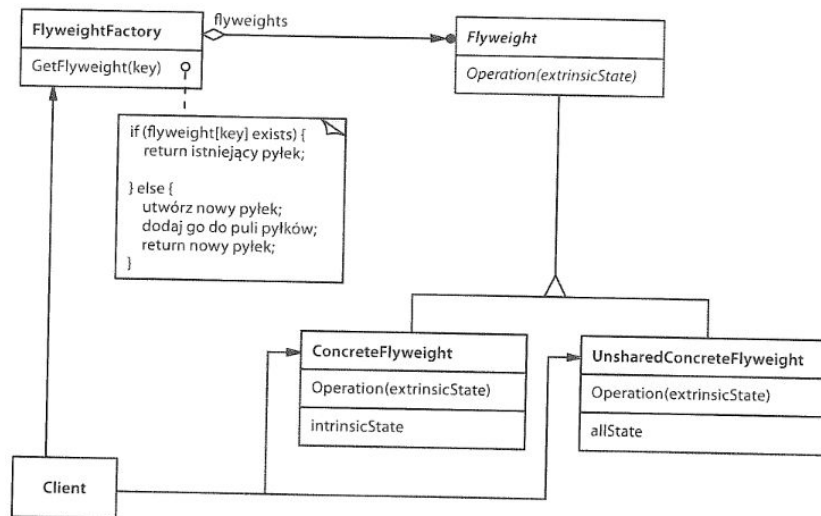
Flyweight:
deklaracja interfejsu do utrzymywania zewnętrznego stanu

ConcreteFlyweight:
umożliwia współużytkowanie, przechowuje stan wewnętrzny

UnsharedConcreteFlyweight:
nie umożliwia współużytkowania, przechowuje cały stan

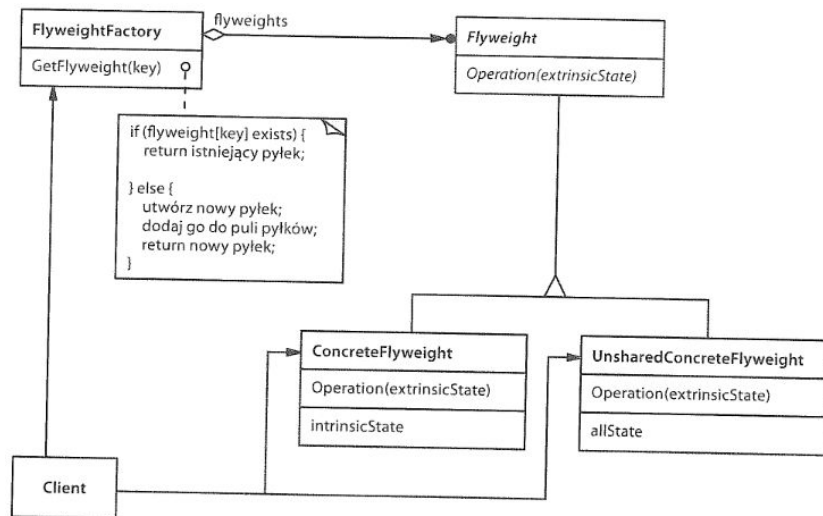
FlyweightFactory:
tworzy obiekty pyłki

Client:
utrzymuje kolekcję pyłków



Pyłek

Oszczędność pamięci



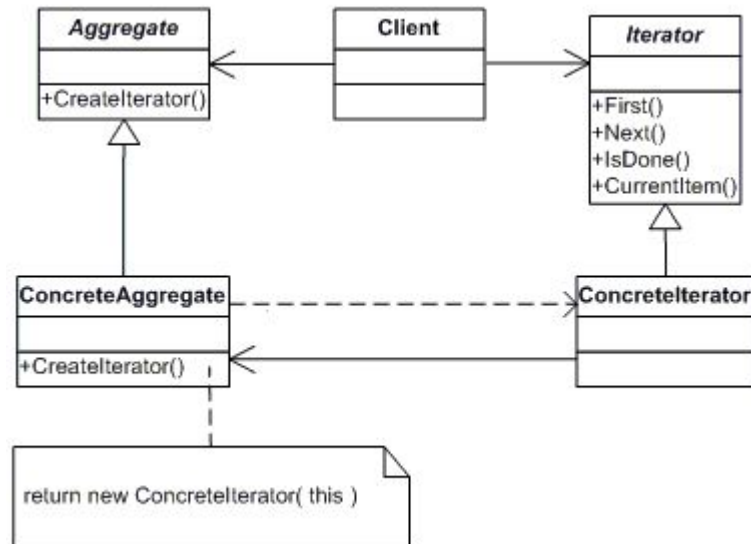
Iterator

Cel: zapewnienie sekwencyjnego dostępu do elementów złożonego obiektu bez ujawniania reprezentacji.

Zastosowanie:
uzyskanie dostępu do agregatu bez ujawniania reprezentacji wewnętrznej

umożliwienie wykorzystania wielu procesów dostępu do agregatu

ujednolicenie interfejsu przeglądania agregatów



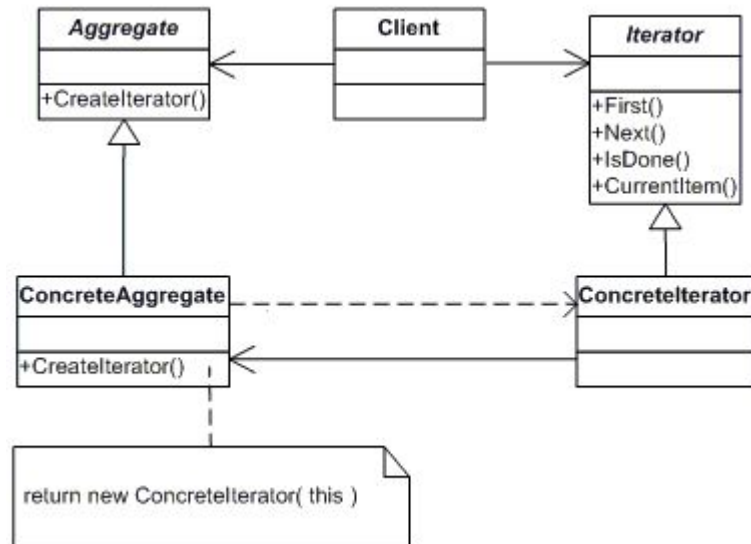
Iterator

Iterator
definicja interfejsu przeglądania agregatu

ConcreteIterator
implementacja interfejsu
śledzenie pozycji

Aggregate
interfejs do tworzenia iteratorów

ConcreteAggregate
Implementacja agregatu

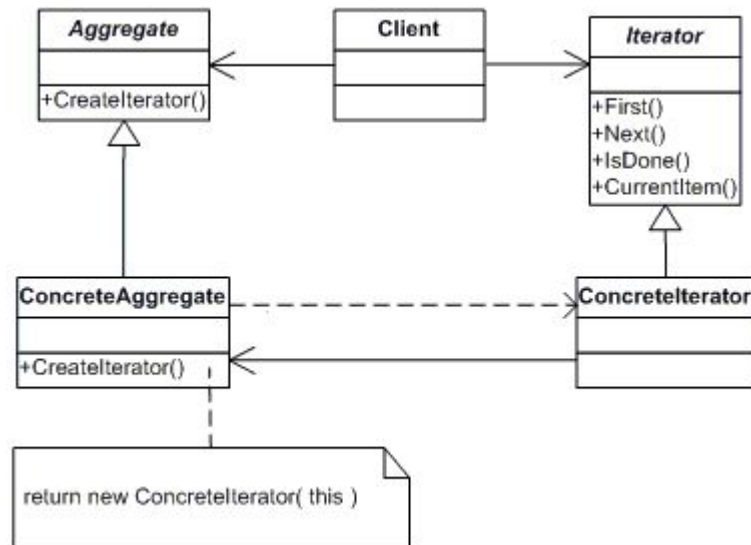


Iterator

Umożliwienie zmiany przeglądania agregatu

Uproszczenie agregatu

Jednoczesne działanie wielu procesów przechodzenia



Łańcuch zobowiązań

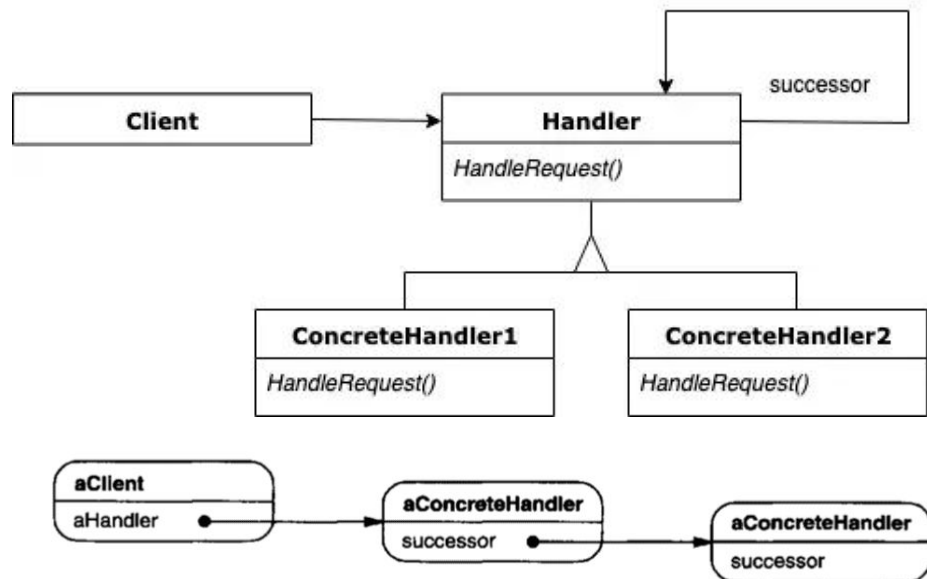
Cel: unikanie wiązania żądania z odbiorcą, łączenie w łańcuch odbiorczy i przekazywanie żądania

Zastosowanie:

gdy więcej niż jeden obiekt obsługuje żądanie, nie wiadomo który

brak określonego odbiorcy

dynamicznie określony zbiór obiektów obsługi



Łańcuch zobowiązań

Handler

interfejs obsługi żądania

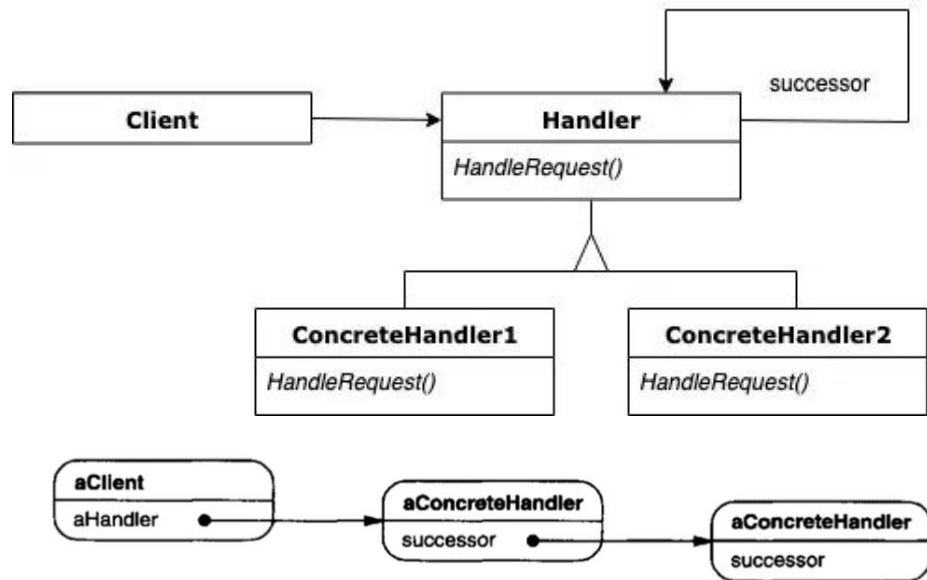
ConcreteHandler

obsługuje żądanie

zyskuje dostęp do następnika

Client

inicjuje żądanie do pierwszego elementu łańcucha

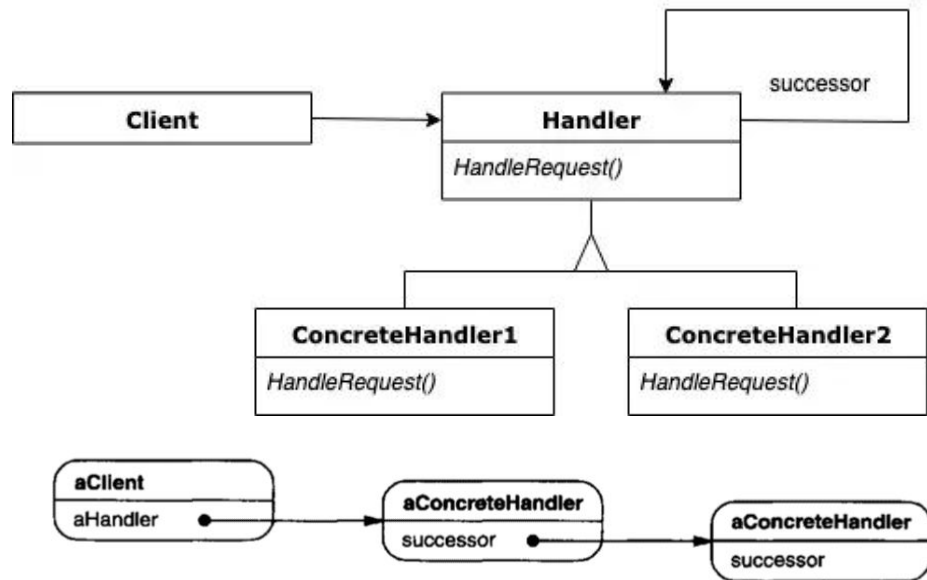


Łańcuch zobowiązań

Zapewnienie mniejszego powiązania

Zwiększenie elastyczności

Brak gwarancji odbioru żądania



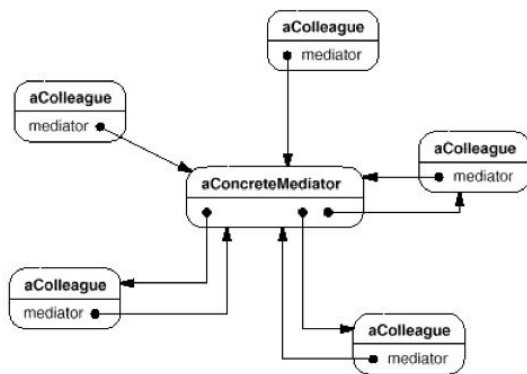
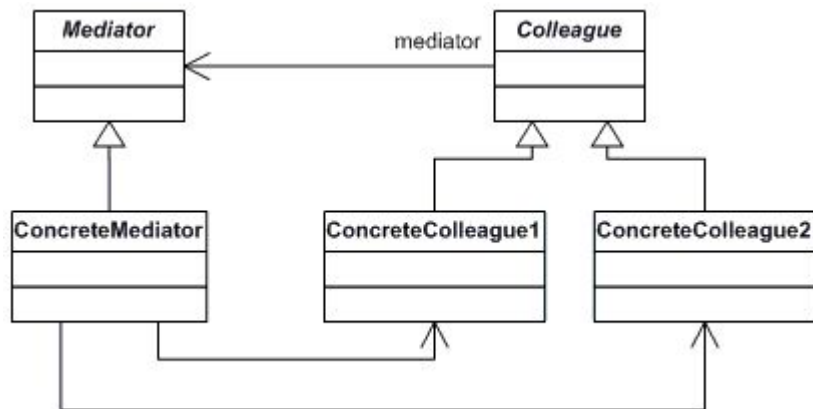
Mediator

Cel: enkapsulacja informacji o interakcji między obiektami

Zastosowanie:
dobrze zdefiniowany lecz skomplikowany sposób komunikacji

trudne ponowne wykorzystanie obiektu

dostosowanie zachowania rozproszonego nie powinno wymagać dodawania podklas

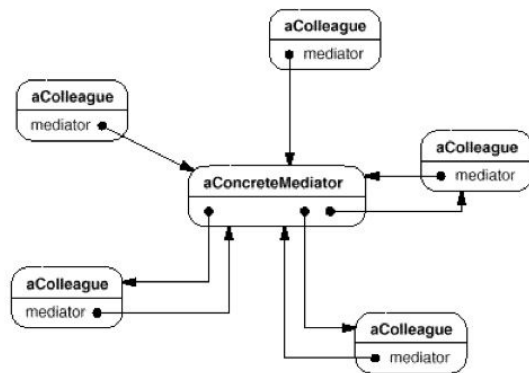
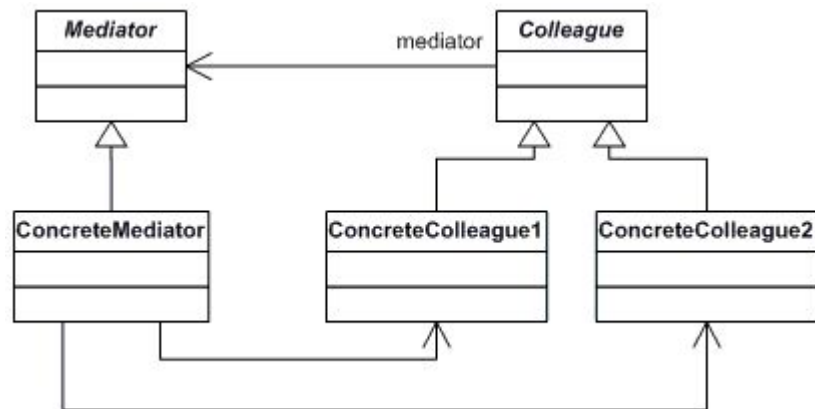


Mediator

Mediator
interfejs komunikacji między obiektami współpracy

ConcreteMediator
koordynacja obiektów współpracujących
zarządzanie powiązaniami

Colleague
obiekt współpracujący
porozumiewa się z Mediatorem



Mediator

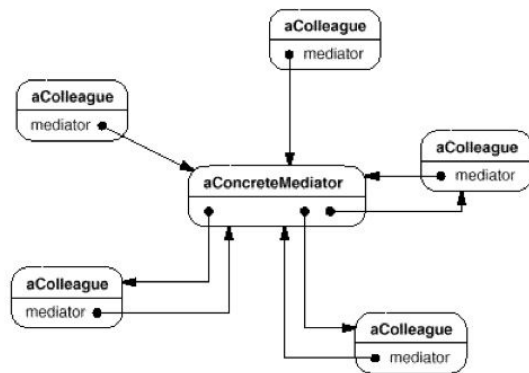
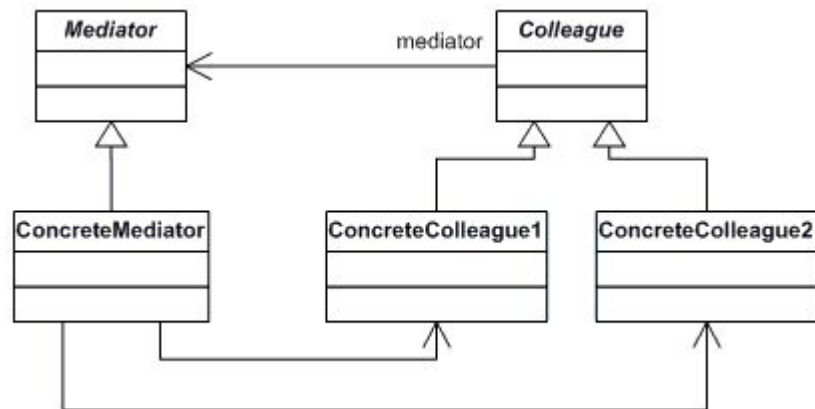
Ograniczenie tworzenia podklas

Rozdzielenie współpracowników

Uproszczenie protokołów obiektów

Ujęcie abstrakcyjnej komunikacji

Centralizacja przepływu



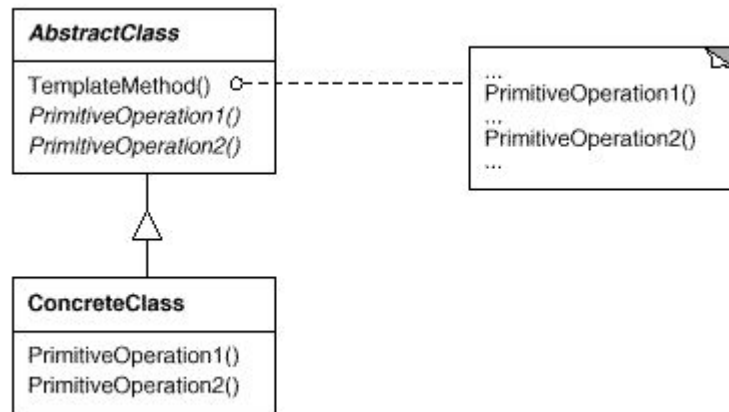
Metoda szablonowa

Cel: określenie szkieletu algorytmu, umożliwienie modyfikacji etapów algorytmu bez zmiany struktury

Zastosowanie:
jednorazowa implementacja niezmiennych części algorytmu

wyodrębnienie wspólnego zachowania klas

kontrolowanie rozszerzania podklas



Metoda szablonowa

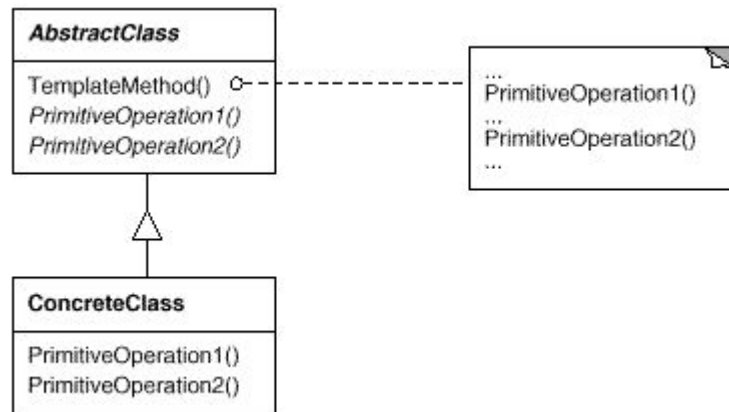
AbstractClass

określenie operacje proste

implementuje metodę szablonową, wywołującą operacje proste

ConcreteClass

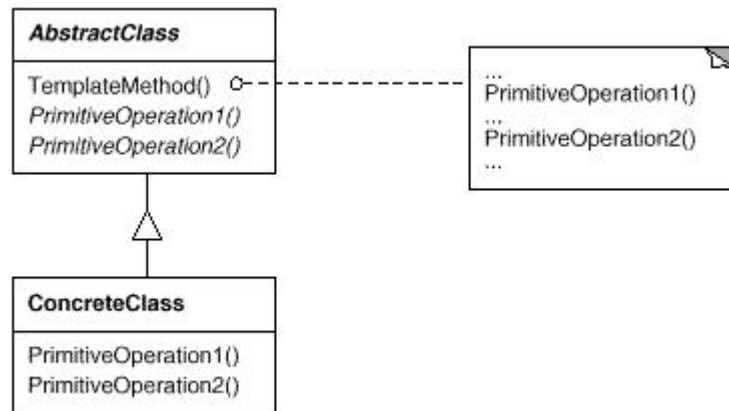
implementacja operacji prostych



Metoda szablonowa

Powtórne wykorzystanie kodu

Odwrócenie struktury sterowania



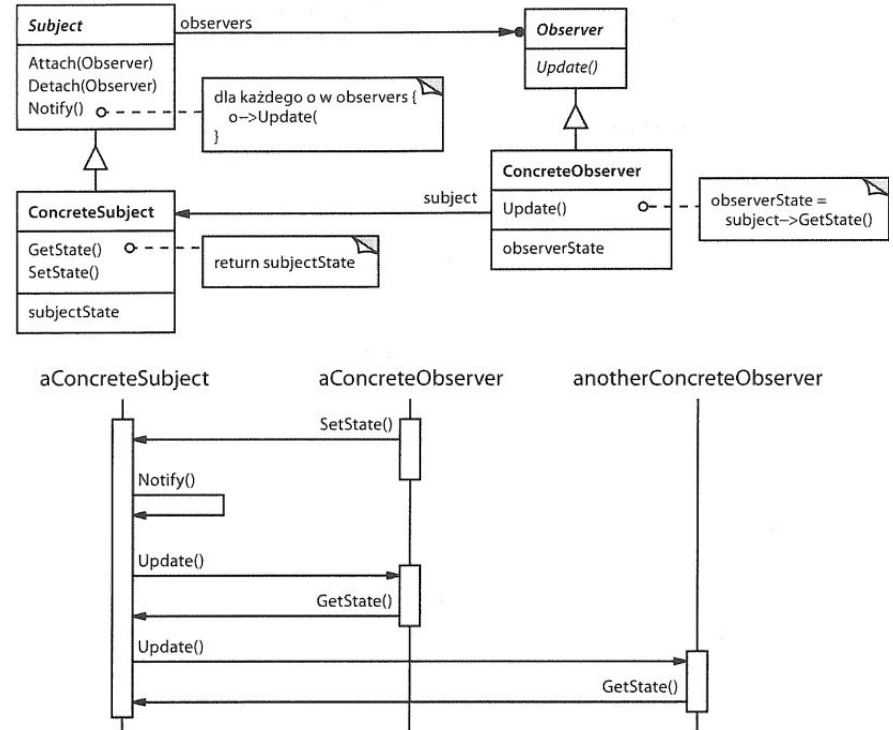
Obserwator

Cel: modelowanie relacji jeden do wielu, model komunikacji broadcast / multicast

Zastosowanie:
dwuaspektowa abstrakcja

konieczność modyfikacji obiektu wynikająca z modyfikacji innego obiektu

brak ścisłego powiązania przy komunikacji



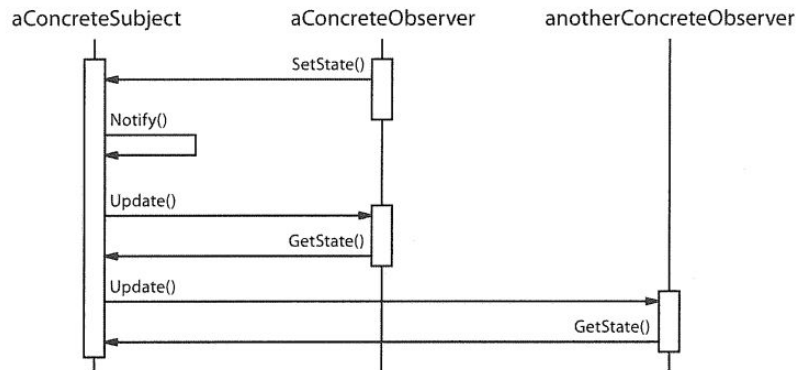
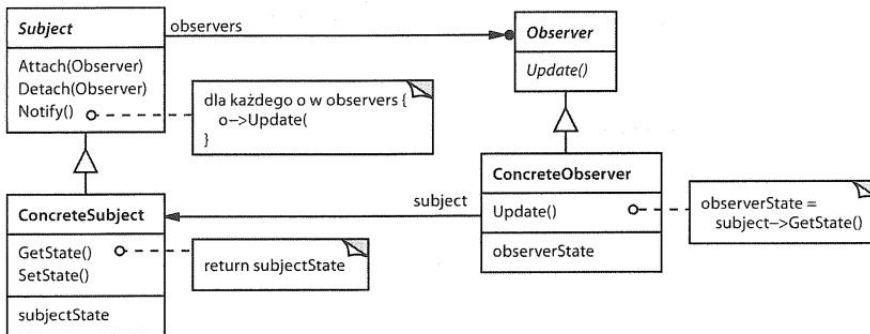
Obserwator

Subject
zna obserwatory
udostępnia dołączanie i odłączanie obserwatorów

Observer
interfejs do powiadamiania obiektów

ConcreteSubject
przechowuje stan istotny dla ConcreteObserver
wysyła powiadomienie po zmianie własnego stanu

ConcreteObserver
przechowuje referencję do ConcreteSubject
przechowuje stan wspólny ze stanem podmiotu
implementacja interfejsu aktualizacji Observer

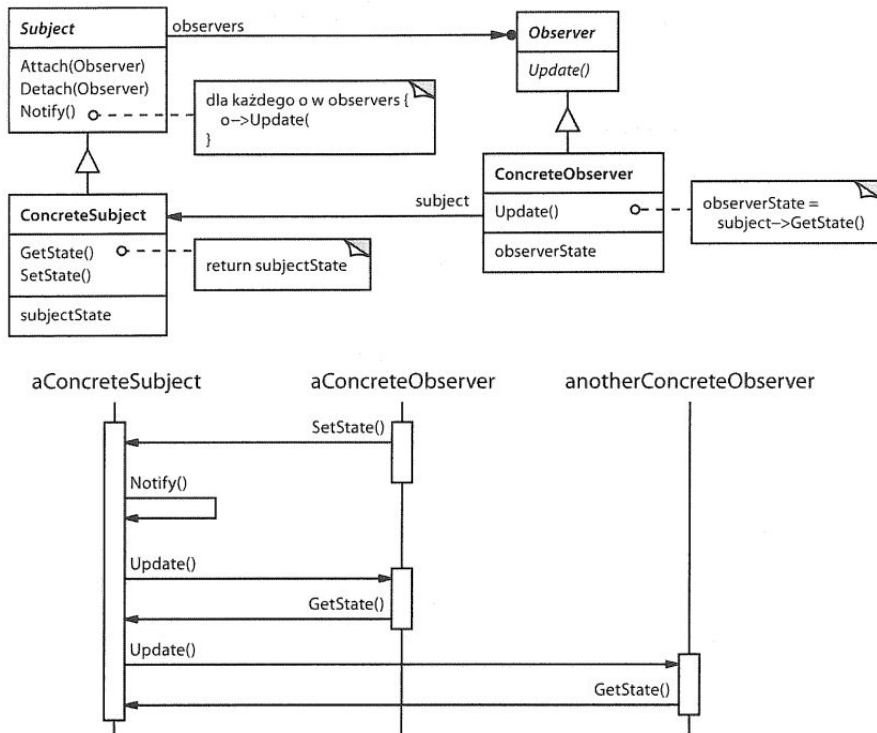


Obserwator

Abstrakcyjne wiązanie między obiektami podmiotu i obserwatora

Obsługa grupowego rozsyłania komunikatów

Ryzyko kaskady powiadomień



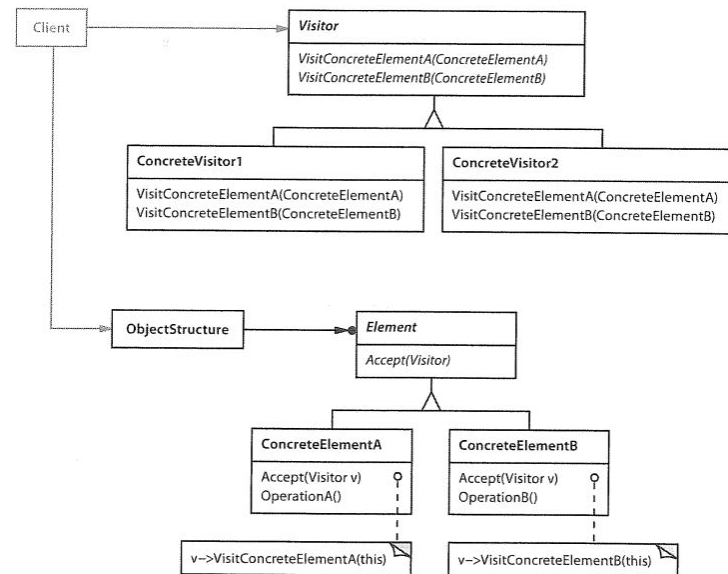
Odwiedzający

Cel: reprezentacja operacji na elementach struktury obiektów

Zastosowanie:
struktura obiektów obejmująca wiele klas o różnych interfejsach, chęć wykonywania operacji zależnych od klas konkretnych

wykonywanie wielu różnych i niepowiązanych operacji

rzadkie zmiany klas przy częstej definicji nowych operacji



Odwiedzający

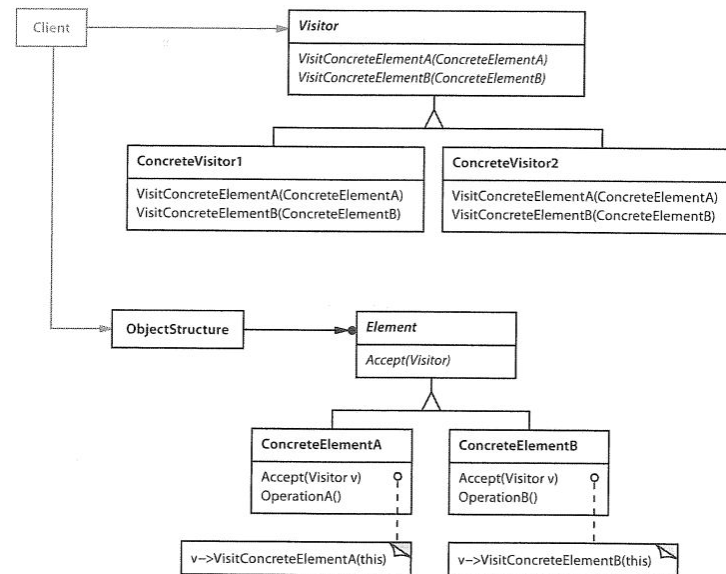
Visitor
deklaracji operacji Visit dla każdej klasy ConcreteElement

ConcreteVisitor
implementacja interfejsu Visitor

Element
definicja operacji Accept

ConcreteElement
implementacja operacji Accept

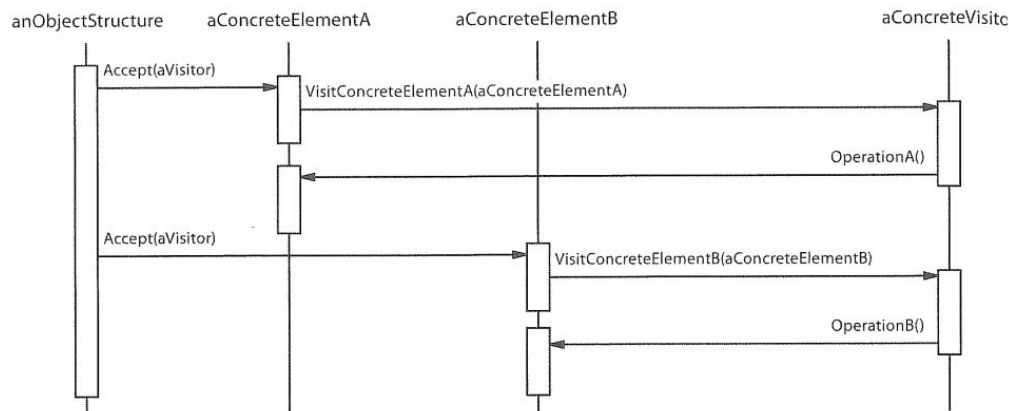
ObjectStructure
przechowuje elementy
kompozyt, lista, etc.



Odwiedzający

Klient tworzy ConcreteVisitor, odwiedza każdy element za pomocą odwiedzającego

Element wywołuje operację Visit... odpowiednią dla jego klasy, przekazując samego siebie jako argument



Odwiedzający

Ułatwienie dodawania nowych operacji

Połączenie operacji powiązanych

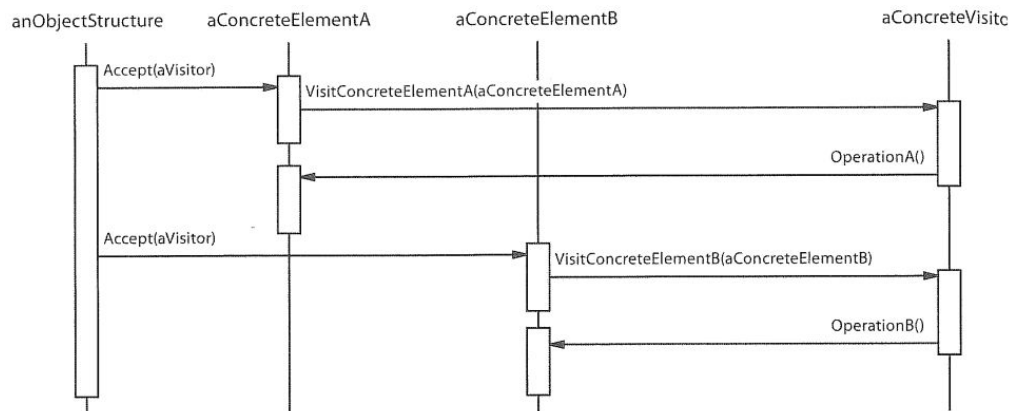
Wyodrębnienie operacji niepowiązanych

Utrudnienie dodawania nowych klas

Możliwość odwiedzania różnych hierarchii klas

Możliwość aktualizacji stanu

Naruszenie enkapsulacji



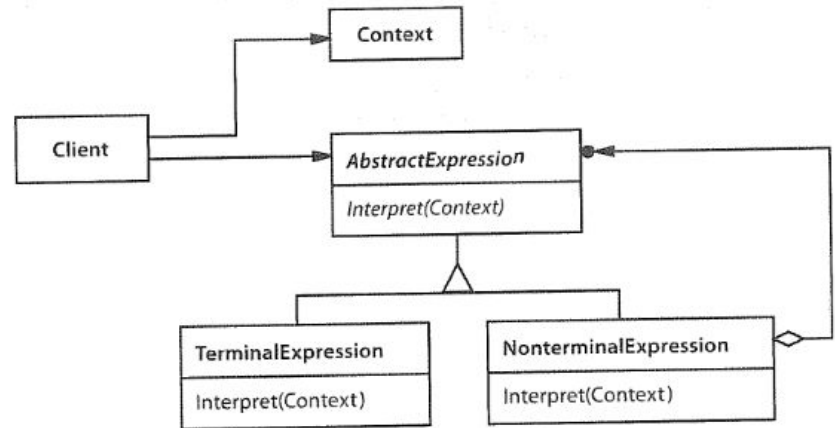
Interpreter

Cel: określenie gramatyki języka, definicja elementu interpretującego zdania w gramatyce

Zastosowanie: istnienie interpretowanego języka możliwego do przedstawienia jako drzewa składni abstrakcyjnej

-prosta gramatyka

-nieistotność wydajności

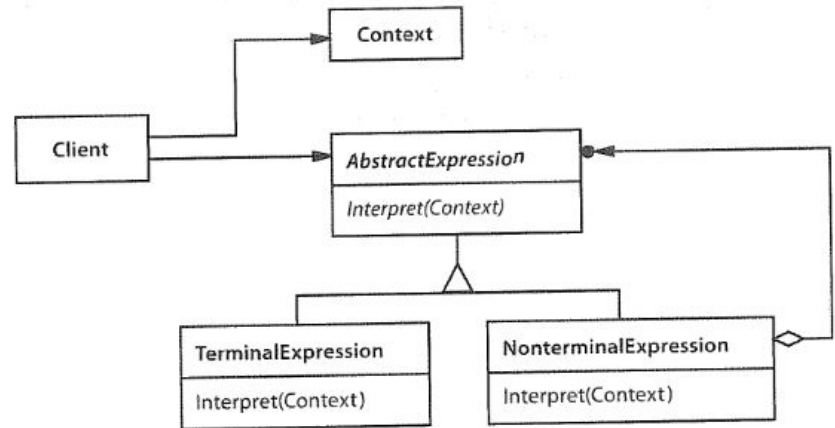


Interpreter

AbstractExpression
deklaracja operacji abstrakcyjnej interpretacji składni

TerminalExpression
implementacja interpretacji symbola końcowego

NonterminalExpression
implementacja operacji Interpret symboli pośrednich



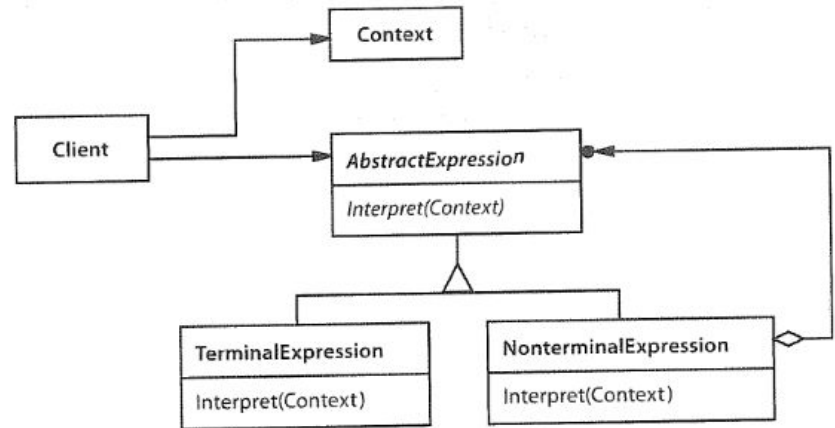
Interpreter

Łatwość modyfikacji i rozszerzania gramatyki

Prosta implementacja gramatyki

Możliwość dodawania nowych interpretacji

Trudność konserwacji złożonych gramatyk

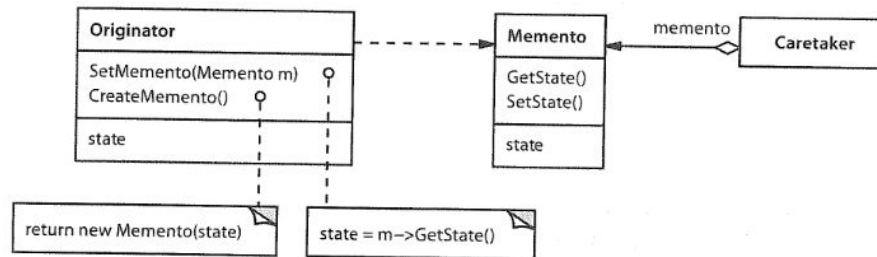


Pamiętka

Cel: zapis stanu wewnętrznego obiektu bez naruszania enkapsulacji celem przywrócenia zapamiętanego stanu

Zastosowanie:
zrzut stanu celem późniejszego odtworzenia

ujawnienie implementacji i naruszenie enkapsulacji przy wykorzystaniu bezpośredniego interfejsu



Pamiętka

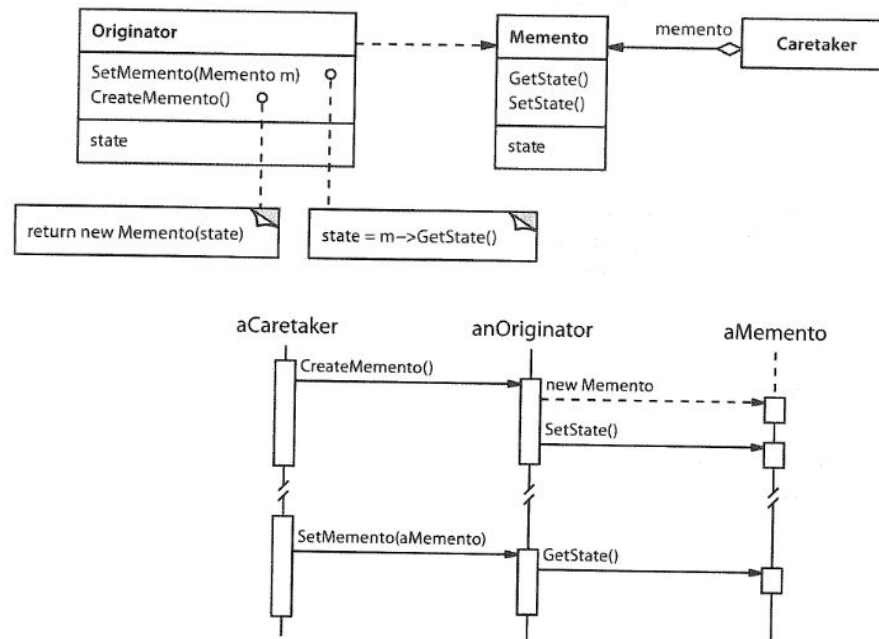
Memento:
przechowuje wewnętrzny stan Originator

chroni dane przez obiekty różne od źródła

Originator:
tworzy pamiętkę

korzysta z pamiętki

Caretaker:
zarządza pamiętkami



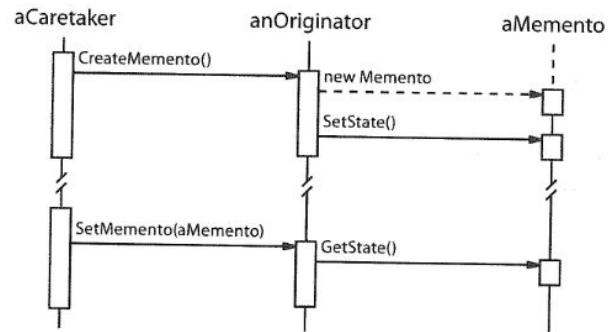
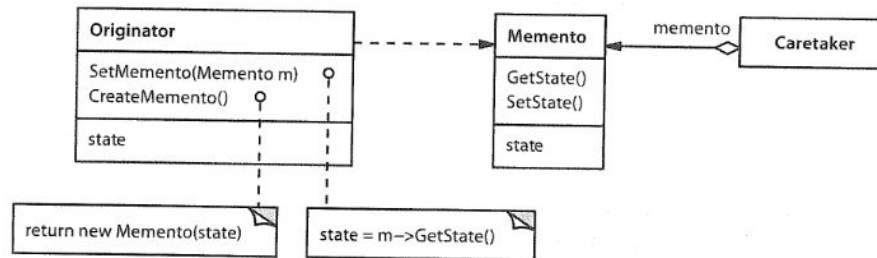
Pamiętka

Zachowanie granic enkapsulacji

Prostota klasy originator

Koszty

Zawężony i pełny interfejs



Polecenie

Command:
deklaracja interfejsu wykonania operacji

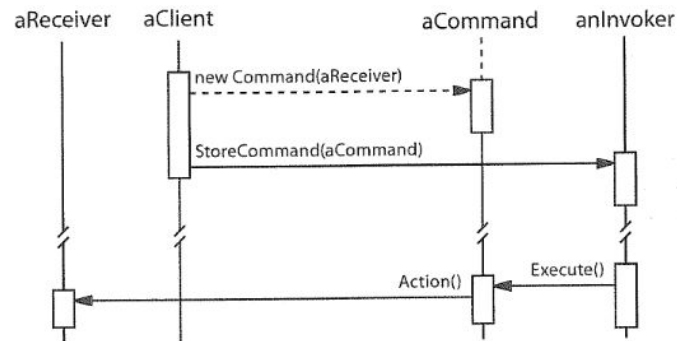
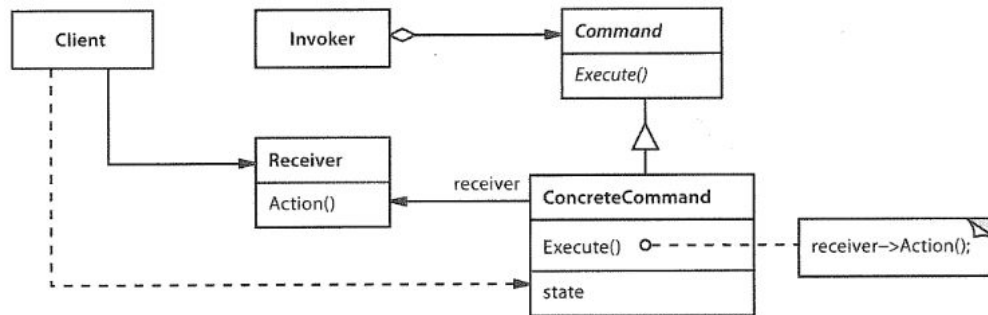
ConcreteCommand:
powiązanie między Receiver i działaniem

implementacja operacji Execute

Client:
tworzenie obiektu ConcreteCommand i określenie powiązanego odbiorcy

Invoker:
żądanie obsługi polecenia

Receiver:
wykoanie ciągu operacji obsługi żądania



Polecenie

Command:
deklaracja interfejsu wykonania operacji

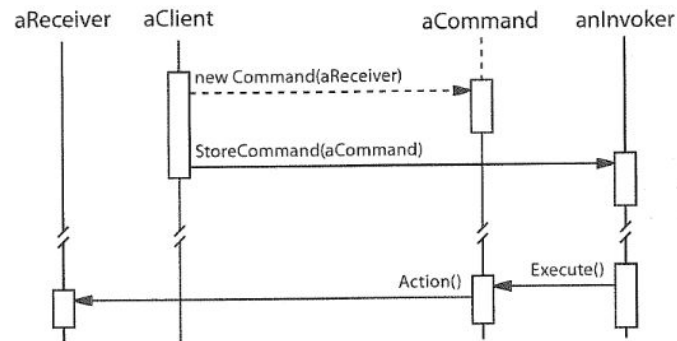
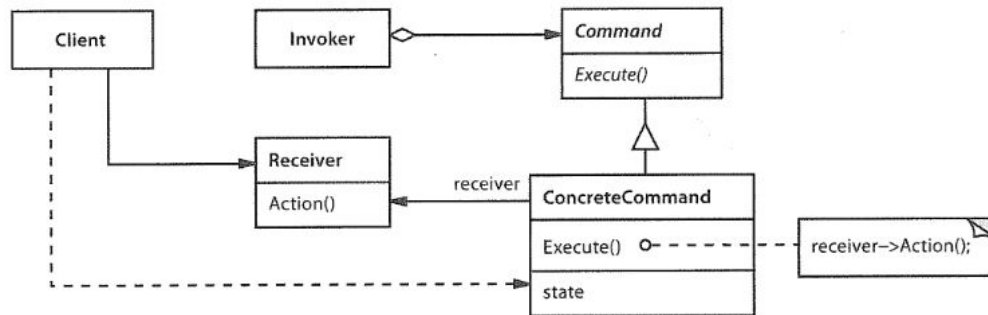
ConcreteCommand:
powiązanie między Receiver i działaniem

implementacja operacji Execute

Client:
tworzenie obiektu ConcreteCommand i określenie powiązanego odbiorcy

Invoker:
żądanie obsługi polecenia

Receiver:
wykoanie ciągu operacji obsługi żądania

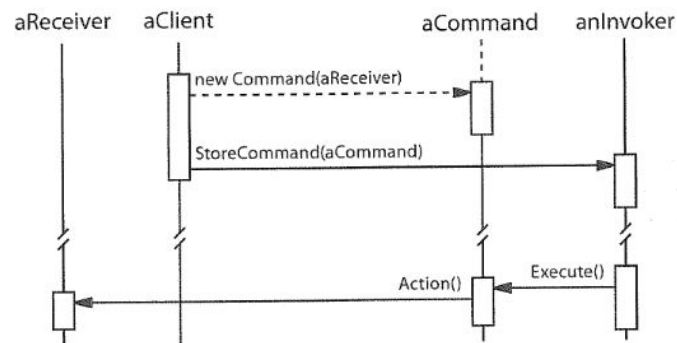
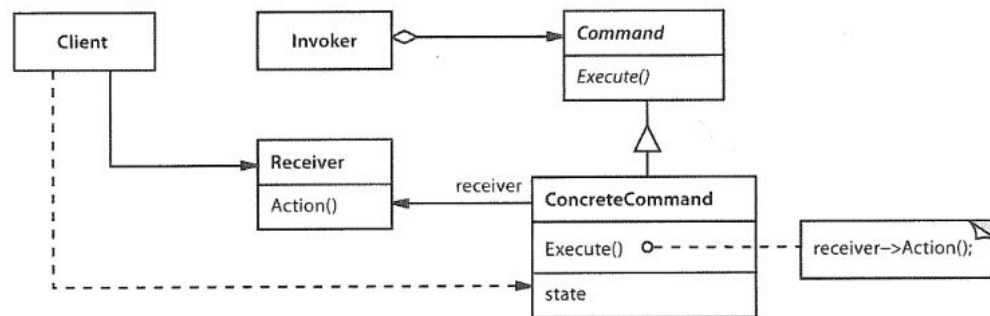


Polecenie

Możność manipulacji poleceniami

Możność łączenia poleceń

Łatwość rozszerzania

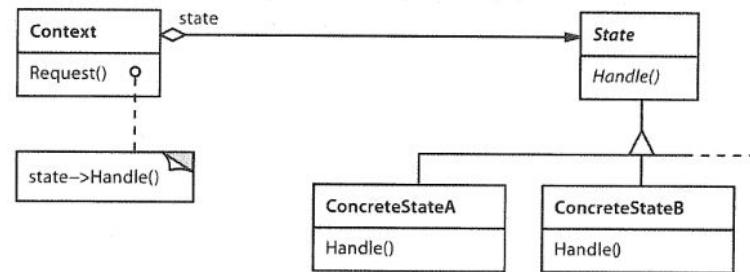


Stan

Cel: Umożliwienie zmiany zachowania w związku ze zmianą stanu

Zastosowanie:
zachowanie obiektu zależy od stanu

istnienie wielu instrukcji warunkowych do sterowania przepływem



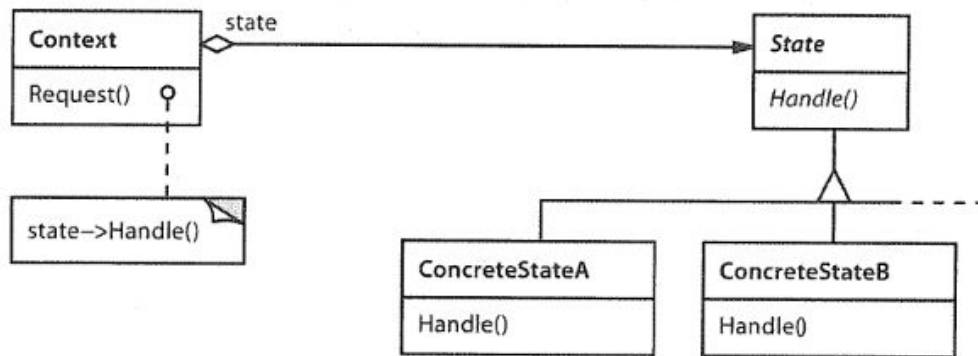
Stan

Context
interfejs udostępniany klientom

definicja bieżącego stanu

State
interfejs kapsułkowania zachowania

ConcreteStateN
implementacja zachowania

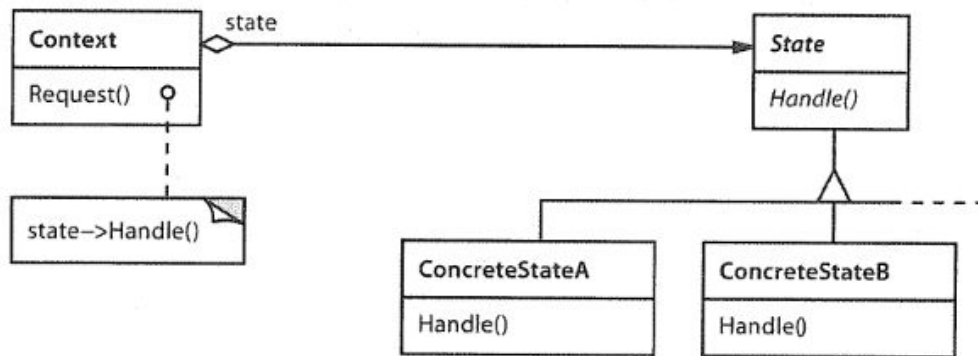


Stan

Grupowanie zachowań

Jawność zmian stanu

Współużytkowanie

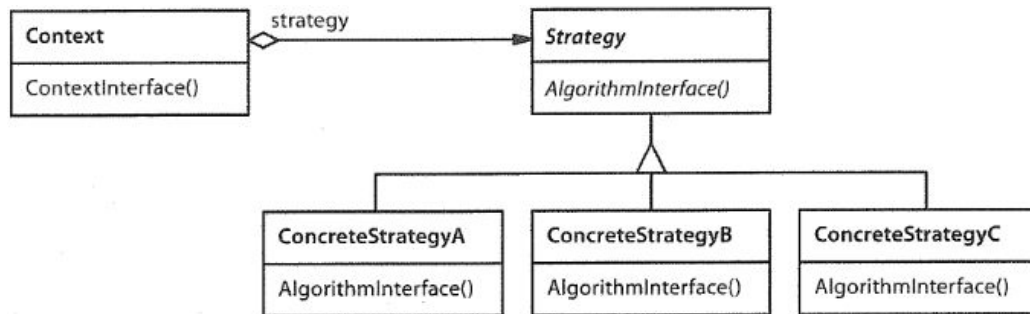


Strategia

Cel: określenie i enkapsulacja rodziny algorytmów

Zastosowanie:
klasy różniące się jedynie zachowaniem
różne wersje algorytmu

sterowanie za pomocą instrukcji warunkowych

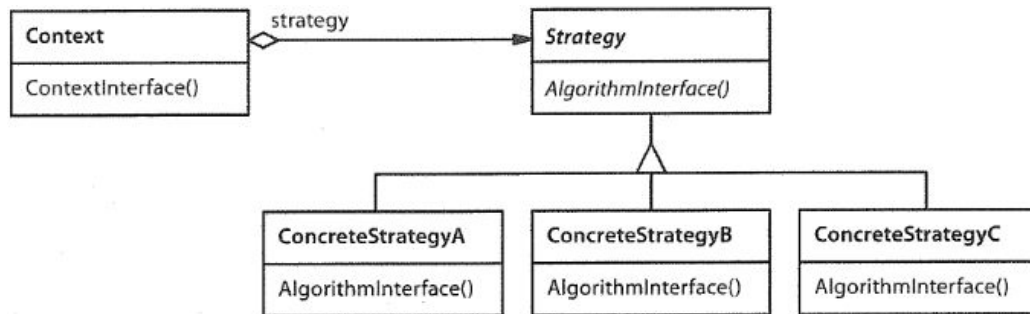


Strategia

Strategy:
wspólny interfejs dla algorytmów

ConcreteStrategy:
implementacja algorytmu

Context:
konfigurowany za pomocą strategii



Strategia

Alternatywa dla dziedziczenia

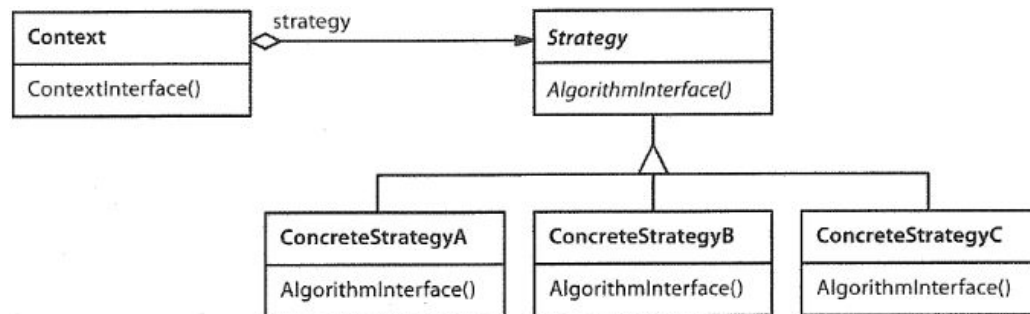
Eliminacja instrukcji warunkowych

Możliwość wyboru implementacji

Konieczność znania strategii przez klientów

Koszt

Zwiększenie liczby obiektów





Słowo końcowe

Wzorce projektowe w postaci zaczerpniętej z literatury przedmiotu:

Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku. Erich Gamma, John Vlissides, Ralph Johnson, Richard Helm

Schematy zgodne z podręcznikiem.