Krzysztof Regulski
AGH, WIMiIP

WZORCE STRUKTURALNE

- PSK - projektowanie systemów komputerowych, notatki w Internecie, Beata Frączek, http://brasil.cel.agh.edu.pl/~09sbfraczek
- http://wazniak.mimuw.edu.pl
- http://www.hillside.net/patterns
- http://www.ii.pw.edu.pl/~ibl/iop2-laboratorium/01/0106.html
- http://www.metal.agh.edu.pl/~banas/IO/SE_W11_12_Projekt_Wzorce.pdf

---

wzorce strukturalne

- opisują sposob konstrukcji struktur obiektowych
- korzystają z dziedziczenia i delegacji
- obejmują: adapter, fasada, dekorator, kompozyt, most, pełnomocnik i pyłek.

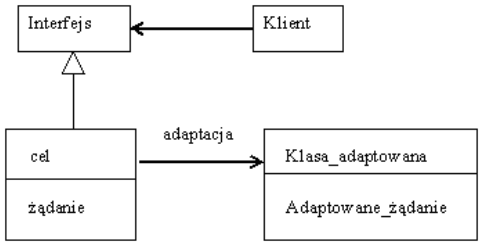
1) Wzorzec projektowy Adapter

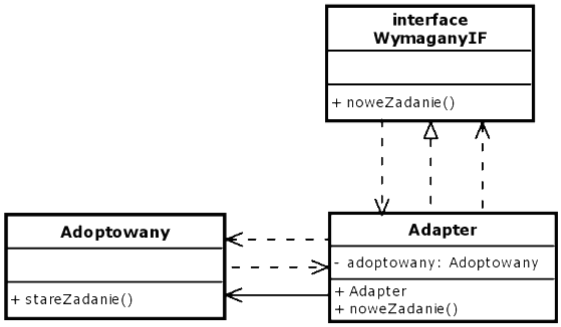
- **Intencja:** Dopasowanie istniejącego obiektu (nad którym programista nie ma kontroli) do określonego sposobu wywołania.
- **Problem**: Obiekt przechowuje potrzebne dane oraz zachowuje się w pożądany sposób, jednak posiada nieodpowiedni interfejs. Często włączamy go do hierarchii dziedziczenia pewnej klasy abstrakcyjnej posiadanej lub definiowanej.
- **Rozwiązanie**: *Adapter* obudowuje obiekt pożądanym interfejsem
- **Uczestnicy i współpracownicy**: *Adapter* dostosowuje interfejs klasy *klasa-adaptowana* tak, by był zgodny z interfejsem klasy bazowej *cel*. Umożliwia to użytkownikowi wykorzystanie obiektu klasy *klasa-adaptowana* oraz instancji klasy *cel*.
- **Konsekwencje**: Zastosowanie wzorca adapter pozwala dopasować istniejące obiekty do tworzonych struktur klas i uniknąć ograniczeń związanych z ich interfejsem.
- **Implementacja**: Zawiera istniejąca klasę w nowej klasie, która posiada wymagany interfejs i wywołuje odpowiednie metody zawieranej klasy
- Adapter można porównać do przejściówki miedzy między różnego rodzaju kablami np. przejściówka miedzy kablem usb a rs232 do podłączenia myszki.
- Adapter zmienia interface klasy B w interface klasy A który rozumie klasa C.
- Alternatywna nazwa wzorca - Wrapper, która oznacza opakowanie, bardzo dobrze opisuje rolę obiektu Adapter: pełnić wobec Klienta rolę otoczki, która umożliwia przetłumaczenie jego żądań na protokół zrozumiały dla faktycznego wykonawcy poleceń.
- Zastosowanie wzorca adapter pozwala dopasować istniejące obiekty do tworzonych struktur klas i uniknąć ograniczeń związanych z ich interfejsem oraz jakiejkolwiek zmiany kodu w klasach.

Zastosowanie:

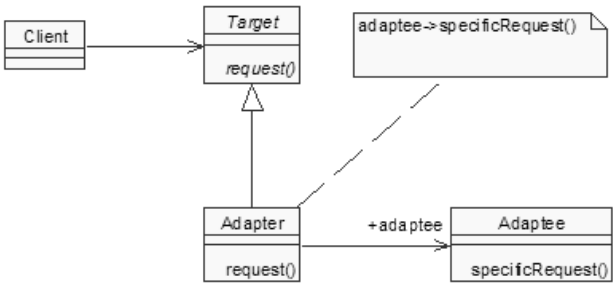
- Gdy klasa nie może być użyta ponieważ ma niekompatybilny interfejs
- gdy nie mamy kodu źródłowego klas nie możemy zmienić ich interfejsu a nawet jeśli mamy kod źródłowy klasy nie powinniśmy raczej zmieniać interfejsu klasy!

Struktura





- Interfejs jest definiowany w abstrakcyjnej klasie *Interfejs* a jego funkcjonalność w podklasach. Metody klasy *cel* mogą dokonywać konwersji typów lub tworzyć nowe algorytmy określające funkcjonalność nie przewidzianą w adaptowanych klasach. Wywołanie metody *klasy-adaptowanej* jest dokonywane przez trawersowanie związku.
- Struktura wzorca składa się z trzech podstawowych klas: Target, Adaptee oraz Adapter. Target jest interfejsem, którego oczekuje klient. Obiektem dostarczającym żądanej przez klienta funkcjonalności, ale niezgodnego pod względem typu, jest Adaptee. Rolą Adaptera, który implementuje typ Target, jest przetłumaczenie wywołania metody należącej do typu Target poprzez wykonanie innej metody (lub grupy metod) w klasie Adaptee. Dzięki temu klient współpracuje z obiektem Adapter o akceptowanym przez siebie interfejsie Target, jednocześnie wykorzystując funkcjonalność dostarczoną przez Adaptee.
- Wzorzec ten posiada także wersję wykorzystującą dziedziczenie w relacji Adapter-Adaptee. Jednak wersja ta ma pewne niedogodności: powiązania między obiektami są ustalane w momencie kompilacji i nie mogą ulec zmianie; ponadto, język programowania musi umożliwiać stosowanie wielokrotnego dziedziczenia lub dziedziczenia i implementacji interfejsu (jak w przypadku języków Java i C#).



Koncepcja

- Koncepcja wzorca ADAPTER'a jest bardzo prosta: piszemy klasę, która posiada wymagany interfejs, a następnie zapewniamy jej komunikację z klasą, która ma inny interfejs. Istnieją dwa sposoby realizacji:

  - poprzez dziedziczenie - z klasy, która ma niezgodny interfejs wywodzimy klasę pochodną i dopisujemy nowe metody tak, by uzyskać wymagany interfejs.
  - poprzez kompozycję - zawieramy pierwotną klasę wewnątrz nowej i tworzymy metody wymaganego interfejsu realizujące wywołania metod klasy wewnętrznej.

2) Wzorzec projektowy Fasada (ang. Facade)

► Intencją omawianego wzorca jest dostarczenie zjednoczonego i uproszczonego interfejsu, zestawu interfejsów z danego podsystemu. Wzorzec Fasady opisuje interfejs wyższego rzędu, który sprawia, że podsystem jest łatwiejszy w użyciu.

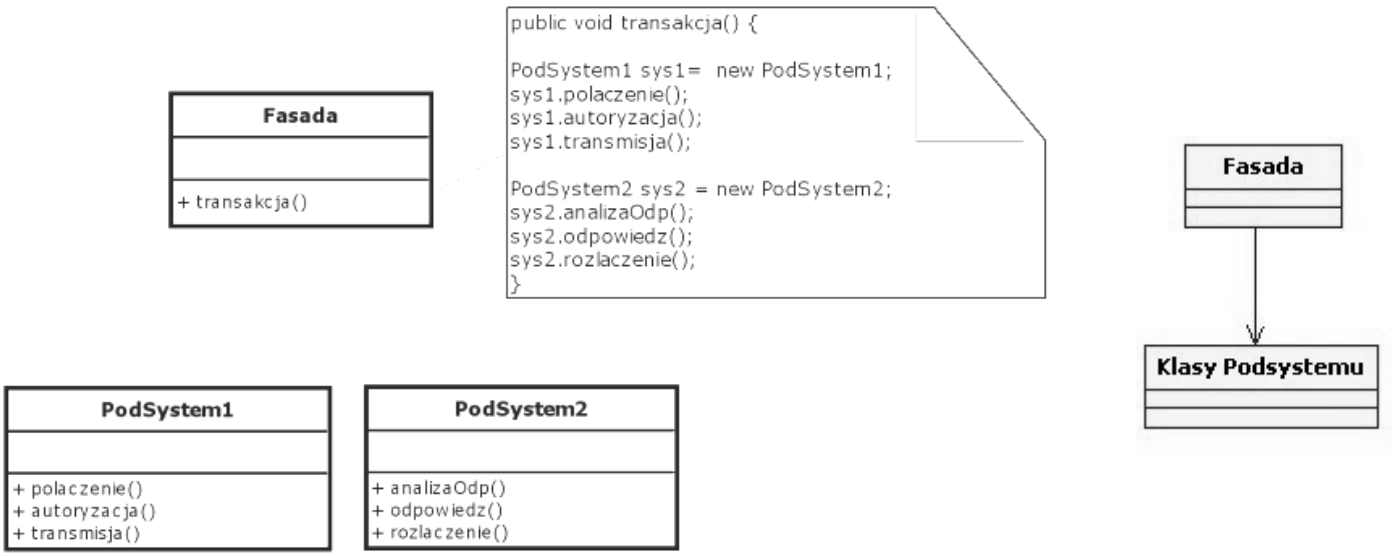
Zalety

- dostarcza prostszy interfejs do rozbudowanego podsystemu bez ograniczania jego funkcjonalności
- osłania klienta od złożoności komponentów podsystemu
- dostarcza "łącznik" pomiędzy podsystem a jego klientów
- ogranicza łączność pomiędzy podsystemami - każdy podsystem używa własnej Fasady i inne części systemu używają wzorca Fasady do komunikowania sie z subsystemami.

Stosowalność

- kiedy potrzebujemy dostarczyć prostszy interfejs do złożonego podsystemu
- kiedy istnieje szereg zależności pomiędzy klientem a klasami implementacji abstrakcji
- kiedy wprowadzenie „warstw” do systemu jest potrzebne albo pożądane

Struktura



Uczestnicy

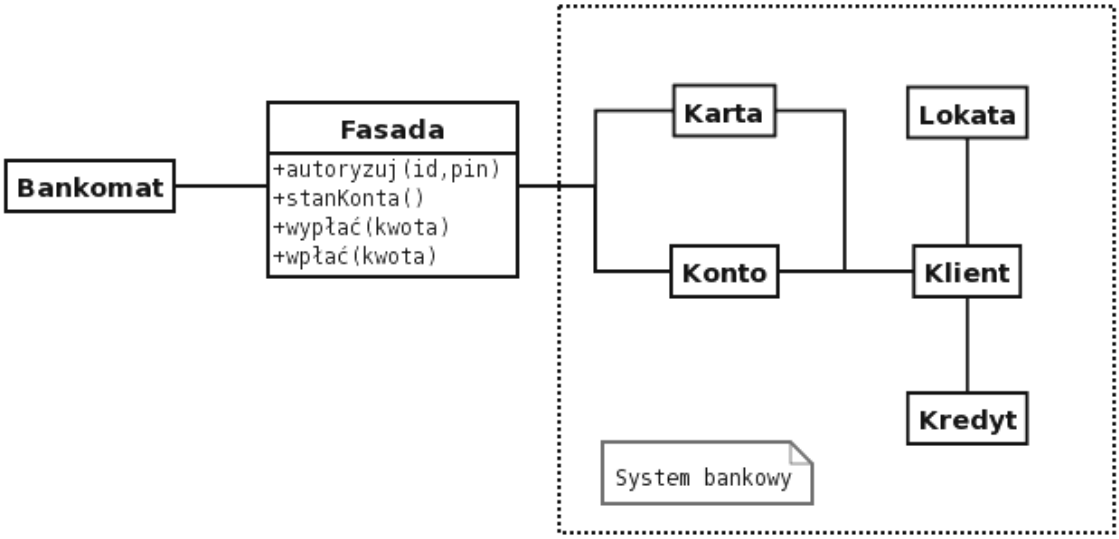
► Fasada - wie jakie klasy podsystemu są odpowiedzialne za spełnienie żądania; przekazuje żądania klienta do odpowiednich obiektów podsystemu.

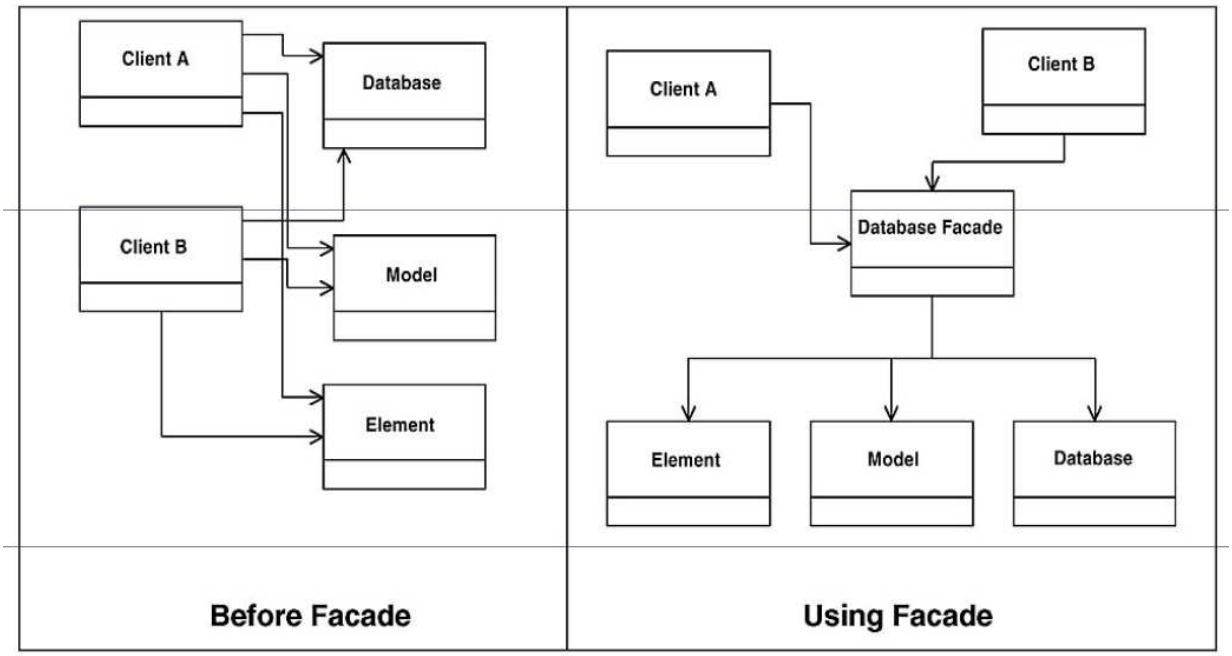
► Klasy podsystemu:
- Implementują funkcje podsystemu;
- wykonują pracę przydzieloną przez obiekt klasy Fasada;
- nic nie wiedzą o fasadzie, to znaczy nie przechowują żadnych odwołań do niej.

Współpraca

- Klienci komunikują się z podsystemem, wysyłając żądania do Fasady, która przekazuje je do odpowiednich obiektów podsystemu. Choć to obiekty podsystemu wykonują właściwą pracę, fasada może być zmuszona realizować swoje zadania, polegające na tłumaczeniu jej interfejsu na interfejsy podsystemu. Klienci wykorzystujący fasadę nie muszą mieć bezpośredniego dostępu do obiektów jej podsystemu.

Przykłady zastosowania





3) Wzorzec projektowy Dekorator

- Jego zadaniem jest rozszerzenie funkcjonalności obiektu poprzez dynamiczne dołączenie dodatkowych zachowań.
- Wzorzec ten pozwala na "dekorowanie" zachowania klasy, czyli zmianę jej funkcjonalności bez potrzeby dziedziczenia.
- Jego działanie można upodobnić do dziedziczenia z tą różnicą, że dziedziczenie rozszerza zachowanie klasy w trakcie kompilacji a dekoratory rozszerzają klasy w czasie działania programu.
- Dekorator zmienia operacje dowolnego Komponentu przez użycie dodatkowego kodu w stosunku do wywoływanej przy tym operacji obiektu dekorowanego.

Zalety

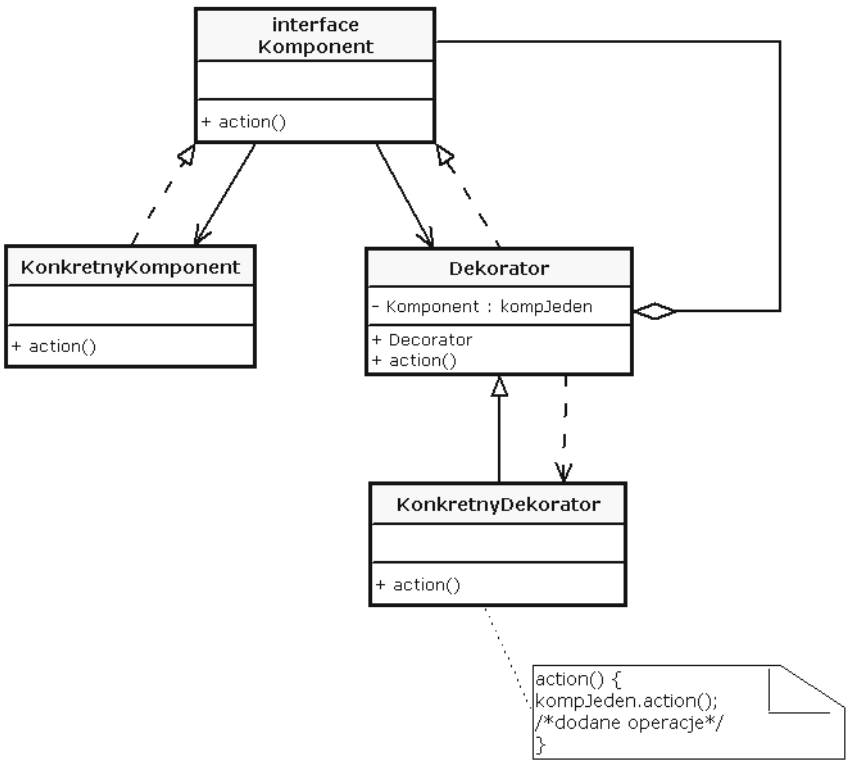
- Większa elastyczność programowanych rozwiązań niż w przypadku dziedziczenia - dynamiczny odpowiednik dziedziczenia obiektu można dowolnie "udekorować" podczas wykonania.
- uproszczenia kodowania poprzez możliwość rozwoju szeregu klas o określonych funkcjach zamiast umieszczania wszystkich zachowań w jednym obiekcie - rozbicie funkcjonalności.

Wady

- Projekty, w których jest używany, charakteryzują się mnogością małych podobnych do siebie obiektów, które różnią się jedynie tym jak są ze sobą połączone. Poznanie i zrozumienie ich w celu modyfikacji może być przez to mocno utrudnione.
- Nie można także korzystać w takich systemach z identyczności obiektów. Dekorator działa jak przezroczysta otoczka, ale z punktu widzenia identyczności obiekt udekorowany nie jest taki sam jak wejściowy.

Stosowalność

- Kiedy chcemy w sposób jawny i dynamiczny dodać nowe zachowania do obiektu bez wpływu na pozostałe obiekty
- Chcemy dodać nowe zachowania do obiektu, wg. których istnieje możliwość że będą zmieniane w przyszłości
- Kiedy rozszerzanie funkcjonalności poprzez podklasy przestaje być praktyczne, np. jeżeli do hierarchii klas musimy dodać nowe podklasy zmieniające zachowania każdego liścia z danej hierarchii. W danej sytuacji lepiej jest stworzyć nową podklasę klasy głównej, która będzie modyfikować odpowiednie zachowanie.

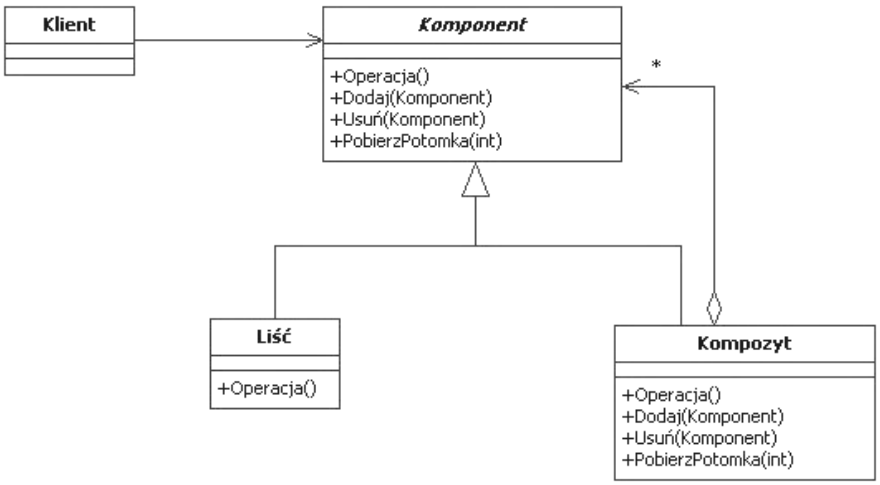


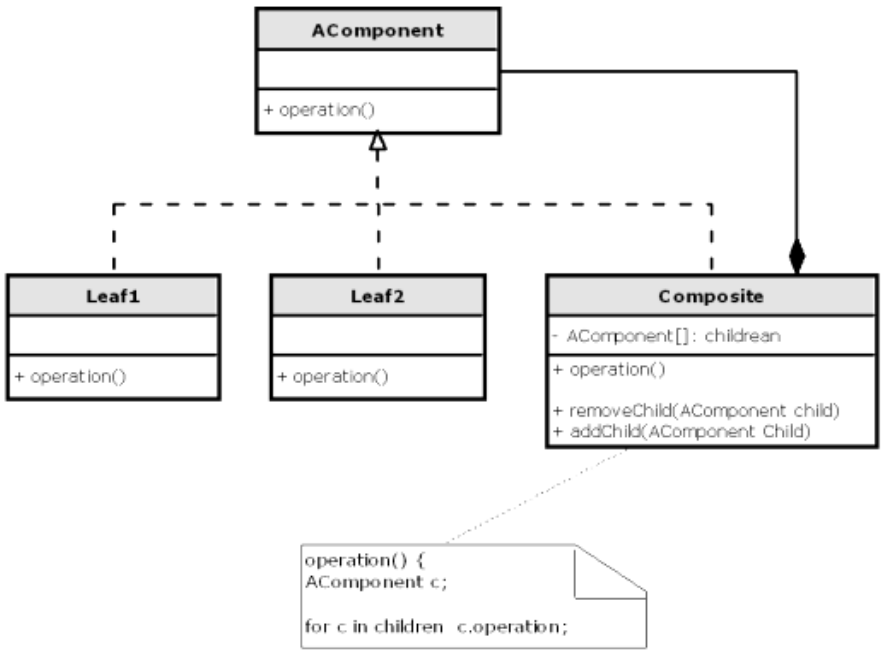
4) Wzorzec projektowy kompozyt (ang. Composite)

- Podczas rozwoju aplikacji często można spotkać komponenty, które mogą być pojedynczymi obiektami lub reprezentować hierarchię obiektów. Wzorzec Kompozyt pozwala klientowi na operowanie w sposób ogólny na obiektach które mogą (lub nie) reprezentować hierarchię obiektów.
- Posługuje się on nazwami uczestników opisującymi ich role w tym wzorcu, co pozwala łatwo stosować go w innych sytuacjach. Klasa Kompozyt jest obiektem złożonym z innych komponentów, interfejs – Komponentem (czyli dowolnym elementem struktury, który można przeszukiwać), natomiast Liść drzewa - obiektem, który nie posiada obiektów zależnych.

Zalety

- klient jednolicie wykonuje operacje na obiekcie złożonym i "prymitywnym"
- łatwo dodawać nowe rodzaje komponentów





Uczestnicy

- Komponent
  - deklaruje interfejs składanych obiektów;
  - implementuje, tam gdzie to możliwe, domyślne zachowanie w wypadku interfejsu wspólnego dla wszystkich klas;
  - definiuje interfejs umożliwiający dostęp i zarządzanie komponentami-dziećmi;
  - (czasami) definiuje interfejs umożliwiający dostęp do rodzica komponentu w strukturze rekurencyjnej i implementuje go, o ile jest to potrzebne.
- Liść
  - reprezentuje obiekty będące liśćmi w składanej strukturze; liść nie ma dzieci;
  - definiuje zachowanie obiektów pierwotnych w strukturze.
- Kompozyt
  - definiuje zachowanie komponentów mających dzieci;
  - przechowuje komponenty będące dziećmi;
  - implementuje operacje z interfejsu Komponentu związane z dziećmi.
- Klient
  - manipuluje obiektami występującymi w strukturze, wykorzystując do tego interfejs komponentu.

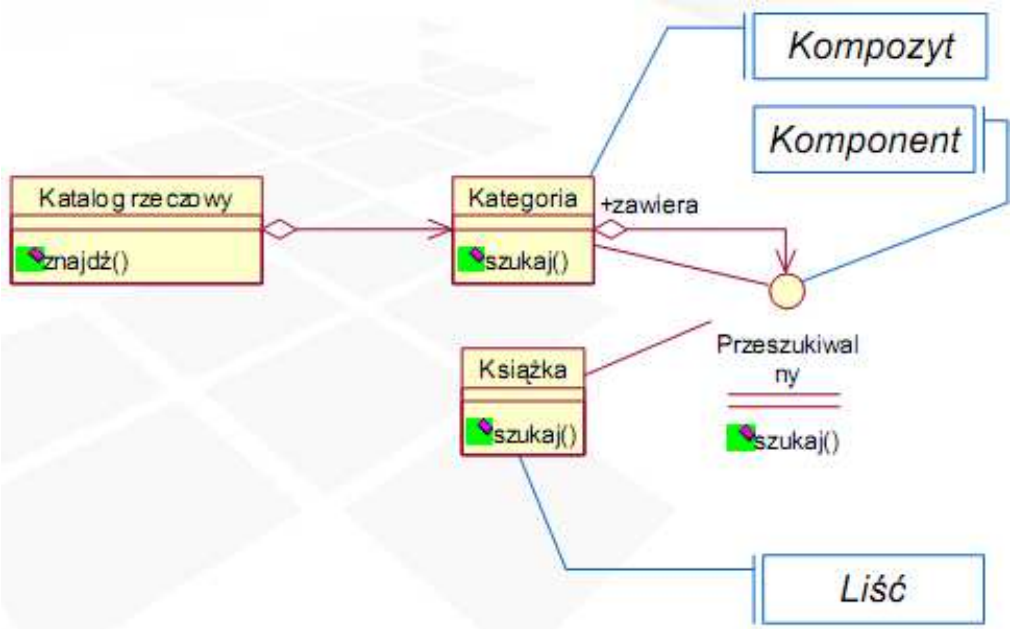
Współpraca

- Klienci używają interfejsu z klasy Komponent w celu komunikowania się z obiektami występującymi w składanej strukturze. Jeśli odbiorca jest Liściem, to żądania są realizowane bezpośrednio. Jeśli natomiast odbiorca jest Kompozytem, to zwykle przekazuje żądania swoim komponentom-dzieciom, być może wykonując przed i/lub po przekazaniu dodatkowe operacje.

Konsekwencje

- Elastyczna definicja struktur drzewiastych - Wzorzec określa metodę konstrukcji hierarchicznych struktur, którymi można zarządzać poprzez jeden węzeł – korzeń. Dzięki temu podstawowe operacje, takie jak wyszukiwanie elementów, nie wymagają żadnej wiedzy o strukturze drzewa.
- Proste dodawanie nowych komponentów - Popularność tego wzorca wynika z oferowanej przez niego możliwości elastycznego zarządzania złożonymi strukturami. Ponadto wszystkie elementy struktury realizują ten sam algorytm, co ułatwia ich testowanie.
- Proste i spójne zarządzanie strukturą o dowolnej liczbie elementów - Mechanizm ten jest jednym z najczęściej wykorzystywanych wzorców projektowych, np. w systemach okienkowych. Strukturę drzewiastą tworzą wówczas składowe okienek: przyciski, etykiety, listy etc. Przesunięcie okienka na ekranie powoduje automatyczne przesunięcie wszystkich jego elementów.

Przykład



5) Wzorzec projektowy Most (ang. Bridge)

Idea

Wzorzec MOST może wydawać się bardzo podobny do wzorca Adapter. Wzorzec ten powstał by oddzielić interfejs od implementacji tak by oba elementy mogły istnieć niezależnie, a co za tym idzie by powstała możliwość wprowadzania zmian do implementacji bez konieczności zmian w kodzie, który korzysta z klasy.

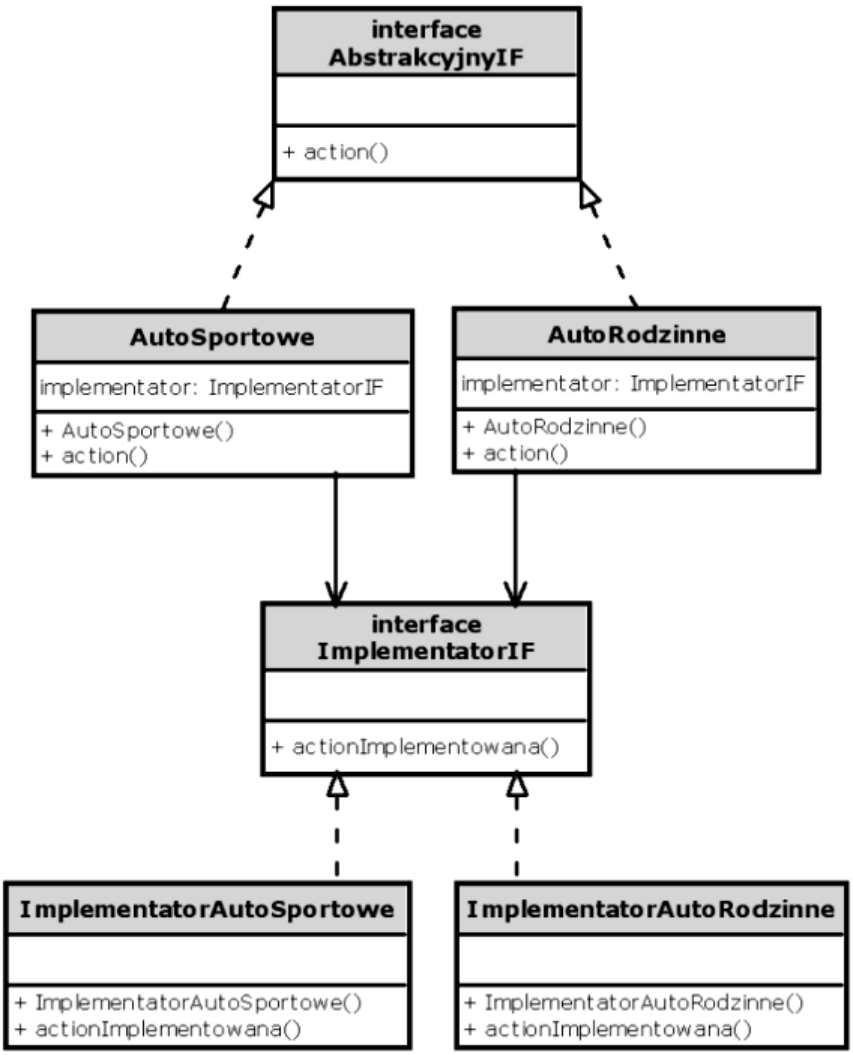
Zalety

- umożliwia odseparowanie implementacji od interfejsu.
- poprawia możliwość rozbudowy klas
- ukrywanie szczegółów implementacyjnych od klienta

Stosowalność

- wymagana jest oddzielność interfejsu od implementacji
- zarówno interfejs jak i implementacja musi zostać rozbudowana poprzez podklasy.
- zmiana implementacyjne nie mogą mieć wpływu na klienta

Struktura



Przykład:

Wyobraźmy sobie abstrakcję, jaką jest figura. Można ją wyszczególnić na np. kwadraty, czy trójkąty, jednak są pewne metody dla każdej figury jak np. rysowanie. Jednak rysowanie może być różne dla różnych bibliotek graficznych czy systemów operacyjnych. Wzorzec mostu pozwala na stworzenie nowych klas, które dostarczają konkretnych implementacji do rysowania. Klasa abstrakcyjna figury dostarcza informacji o figurze (np. wielkość), podczas gdy implementacja dostarcza interfejs do rysowania.

6) Wzorzec projektowy Pełnomocnik (ang. Proxy)

Stosowalność

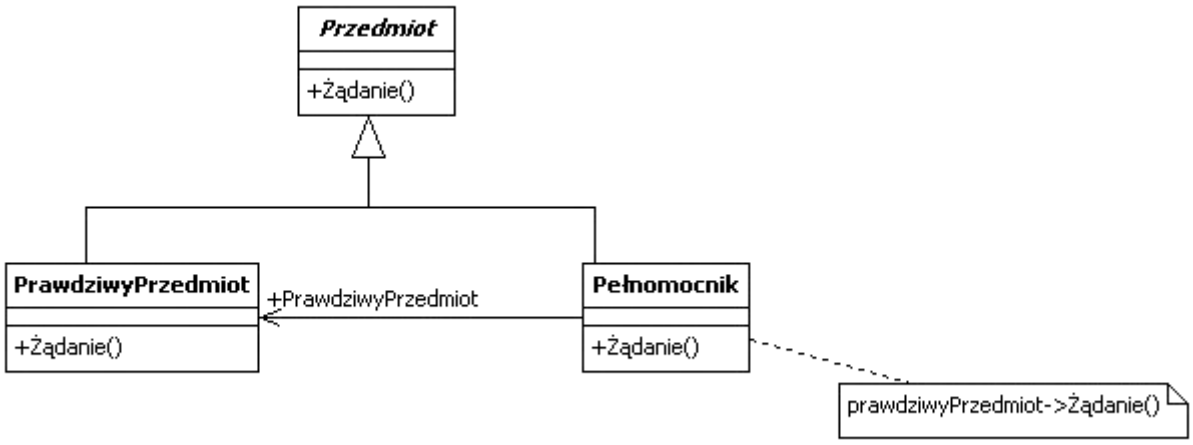
Pełnomocnik ma zastosowanie zawsze wtedy, kiedy potrzeba bardziej uniwersalnego lub bardziej wyrafinowanego odwołania do obiektu niż zwykły wskaźnik. Celem jest utworzenie obiektu zastępującego inny obiekt. Stosowany jest w celu kontrolowanego tworzenia na żądanie kosztownych obiektów oraz kontroli dostępu do nich.

Rodzaje

Istnieją cztery rodzaje tego wzorca, które jednocześnie definiują sytuacje, w których może zostać użyty:

- wirtualny - przechowuje obiekty, których utworzenie jest kosztowne; tworzy je na żądanie
- ochraniający - kontroluje dostęp do obiektu sprawdzając, czy obiekt wywołujący ma odpowiednie prawa do obiektu wywoływanego
- zdalny - czasami nazywany ambasadorem; reprezentuje obiekty znajdujące się w innej przestrzeni adresowej
- sprytne odwołanie - czasami nazywany sprytnym wskaźnikiem; pozwala na wykonanie dodatkowych akcji podczas dostępu do obiektu, takich jak: zliczanie referencji do obiektu czy ładowanie obiektu do pamięci

Struktura



Uczestnicy

- Pełnomocnik
  - przechowuje odwołanie, które umożliwia mu dostęp do prawdziwego przedmiotu; może odwoływać się do przedmiotu, jeśli interfejsy Przedmiotu i prawdziwegoPrzedmiotu są takie same;
  - zapewnia taki sam interfejs jak interfejs Przedmiotu, tak że Pełnomocnik może być zastąpiony przez Przedmiot;
  - kontroluje dostęp do prawdziwego przedmiotu i może być odpowiedzialny za tworzenie oraz usuwanie tego przedmiotu;

inne obowiązki zależą od rodzaju pełnomocnika:

- pełnomocnicy zdalni są odpowiedzialni za kodowanie żądań i ich argumentów oraz za wysyłanie zakodowanych żądań do rzeczywistych przedmiotów w innej przestrzeni adresowej;
- pełnomocnicy wirtualni mogą przechowywać w pamięci podręcznej dodatkowe informacje o rzeczywistym przedmiocie, dzięki czemu mogą odłożyć na później uzyskanie dostępu do niego;
- pełnomocnicy ochraniający sprawdzają, czy wywołujący ma zezwolenie na dostęp, wymagane na spełnienie danego żądania.

► Przedmiot - definiuje wspólny interfejs dla PrawdziwegoPrzedmiotu i Pełnomocnika, tak żeby Pełnomocnik mógł być używany wszędzie tam, gdzie jest oczekiwany PrawdziwyPrzedmiot.

► PrawdziwyPrzedmiot - definiuje rzeczywisty obiekt reprezentowany przez pełnomocnika.

Współpraca

► Pełnomocnik, jeśli trzeba przekazuje żądania do PrawdziwegoPrzedmiotu (w zależności od rodzaju pełnomocnika).

7) Wzorzec projektowy Pyłek (Waga piórkowa, ang. Flyweight)

Idea

► Istotą wzorca jest podział danych przechowywanych w Obiektach na dane wewnętrzne (współdzielone) i zewnętrzne (unikatowe dla każdego obiektu). Dane wewnętrzne nie są modyfikowane przy inicjacji obiektu, natomiast dane zewnętrzne są dostarczane dla każdego obiektu z zewnątrz przed przekazaniem obiektu Klientowi. celem jest zmniejszenie wykorzystania pamięci poprzez efektywną obsługę wielu małych obiektów za pomocą współdzielenia.

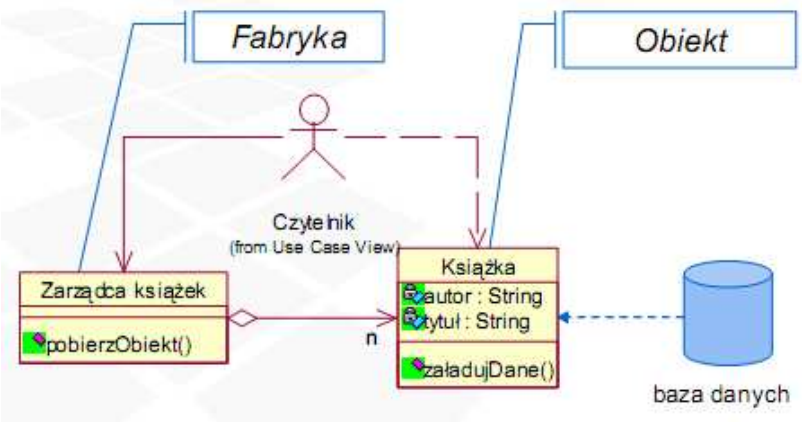
Zalety

► Niektórych programów nie da się napisać w sposób obiektowy bez stosowania wagi piórkowej. W przypadku użycia puli obiektów wagi piórkowej można w większości języków porównywać za pomocą standardowego operatora porównywania ==

Wady

► Jeśli obcy stan jest składowany w kontenerze, dostęp do niego jest możliwy tylko przez kontener, z drugiej strony gdy stan obcy jest wyznaczany na bieżąco, operacje mogą ulec spowolnieniu. Obiekty wagi piórkowej komplikują kod uniemożliwiając często jego konserwację i zwiększają jego rozmiar.

Struktura



Uczestnicy

- Obiekt
  - podlega współdzieleniu między klientów
  - definiuje interfejs do przyjmowania i odtwarzania
  - stanu zewnętrznego obiektu
  - przechowuje stan wewnętrzny (współdzielony)
  - jest niezależny od kontekstu (z wyjątkiem stanu
  - zewnętrznego)
- Fabryka obiektów - tworzy i przechowuje obiekty

Przykład:

- Klasycznym przykładem użycia wzorca wagi piórkowej jest struktura danych dla graficznej reprezentacji znaków w procesorach tekstu:
- Bez użycia tego wzorca, dla każdego znaku w dokumencie przechowywane by były kształty znaku w czcionce, rozmiary czcionek i inne dane używane w formatowaniu tekstu. Wtedy zużycie pamięci dla całego dokumentu było by ogromne.
- Zamiast tego dla każdego znaku można zastosować referencje do obiektu typu czcionka (waga piórkowa) udostępnianego przez każdą instancję tego samego znaku w dokumencie

Konsekwencje

- Zmniejszenie wymagań pamięciowych programu
  - zmniejszenie ogólnej liczby obiektów
  - zmniejszenie rozmiaru stanu obiektów
  - stan zewnętrzny może być przechowywany lub wyliczany
- Wzrost złożoności obliczeniowej
  - dodatkowy nakład na zarządzanie stanem zewnętrznym
- Zastosowanie tego wzorca pozwala na znaczne oszczędności pamięci, szczególnie w aplikacjach korzystających z dużej liczby instancji tego samego typu. Z jednej strony ulega zmniejszeniu ogólna liczba utworzonych obiektów, a z drugiej - rozmiar ich stanów wewnętrznych. Oczywiście, nie uwzględnia to kosztu przechowywania stanu zewnętrznego, jednak w pewnych sytuacjach może on być obliczony, a ponadto nie wymaga on tworzenia i usuwania obiektów, co stanowi główny problem w tego typu aplikacjach.