WZORCE PROJEKTOWE

TABLICE INFORMATYCZNE • Daniel Krasnokucki

Helion

Wzorzec opisuje problem, który powtarza się wielokrotnie w danym środowisku, oraz podaje istotę jego rozwiązania w taki sposób, aby można było je zastosować miliony razy bez potrzeby powtarzania tej samej pracy.

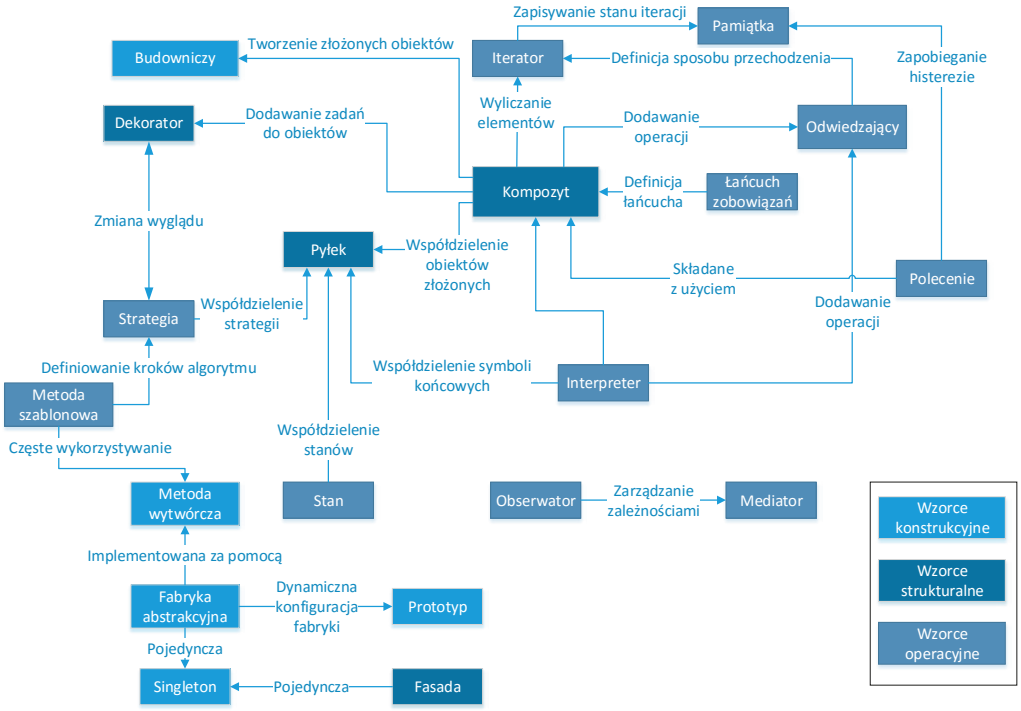
Christopher Alexander „A pattern language", 1977

WPROWADZENIE

Tablice pomogą Ci szybko przypomnieć sobie poszczególne wzorce projektowe oraz ich zastosowanie. Tablice napisane są w oparciu o książkę *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku* i opisują wzorce skatalogowane przez Gang Czworga.
Zalety i wady poszczególnych wzorców mogą być wywnioskowane z ich opisów i podanego przykładowego zastosowania, nie oceniałem ich jawnie, bo wszystko zawsze zależy od projektu, kodu i programisty. Pamiętać należy także, że źle zastosowany wzorzec może przekształcić się w antywzorzec.

Relacje pomiędzy wzorcami

Obok przedstawiono zależności pomiędzy wzorcami opisanymi w niniejszych tablicach.



WZORCE KONSTRUKCYJNE (KREACYJNE)

Pozwalają w sposób abstrakcyjny tworzyć i konfigurować obiekty w celu ich wielokrotnego użycia i zachowania niezależności systemu od sposobu ich tworzenia.

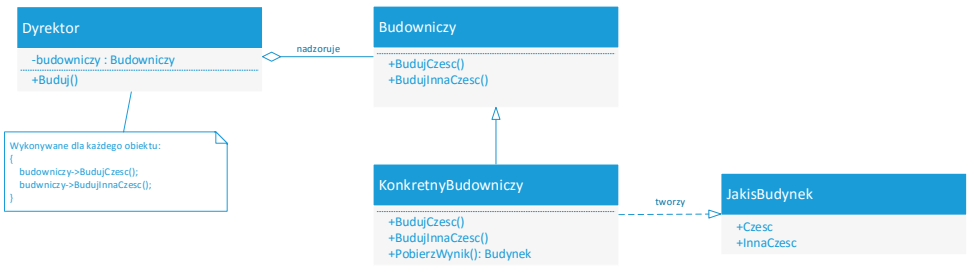
Budowniczy

PRZEZNACZENIE

Wzorzec ten umożliwia tworzenie wielu takich samych obiektów o różnej konfiguracji — oddziela tworzenie obiektów od ich reprezentacji. Za konfigurację poszczególnych obiektów są odpowiedzialne wyspecjalizowane klasy, które implementują interfejs podstawowego obiektu Builder.

IMPLEMENTACJA

1. Tworzymy instancję klasy KonkretnyBudowniczy, która stworzy oczekiwany przez nas rodzaj obiektu.
2. Tworzymy instancję klasy Dyrektor i przekazujemy do niej referencję budowniczego.
3. Wywołujemy metodę konstrukcji budynku w powyższym obiekcie (dyrektor), która zapewni użycie odpowiedniej sekwencji metod budowniczego (dyrektor zleca wykonanie budowniczemu według określonych zasad).
4. Po stworzeniu budynku odbieramy (pobieramy) go od naszego budowniczego.



PRZYKŁADY ZASTOSOWANIA

- tworzenie węzłów XML-a lub HTML-a zależnych od działania programu;
- konwertowanie tekstu, zdjęć, muzyki, wideo — algorytm odczytujący dane będzie niezależny od zapisującego, a dane wyjściowe mogą być tworzone przez wiele podobnych obiektów w zależności od formatu.

Fabryka abstrakcyjna

PRZEZNACZENIE

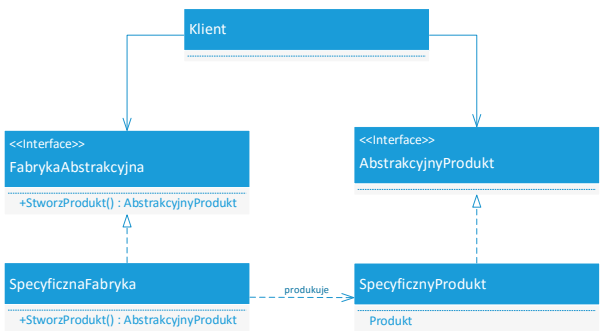
Umożliwia tworzenie rodzin zależnych lub spokrewnionych obiektów w sposób abstrakcyjny, bez opisywania konkretnych klas.

IMPLEMENTACJA

1. Przy implementacji wykorzystywana jest kompozycja.
2. Fabryka zwraca całą rodzinę powiązanych ze sobą obiektów. Zazwyczaj do tworzenia obiektów wykorzystywana jest metoda wytwórcza.

PRZYKŁADY ZASTOSOWANIA

- utworzenie rodziny implementacji różnych interfejsów, która będzie odpowiadała za komunikację z dostępnym API;
- implementacja wyświetlania tekstu w różnych typach plików, takich jak pdf, doc, html itp.
- obsługa różnych typów baz danych.

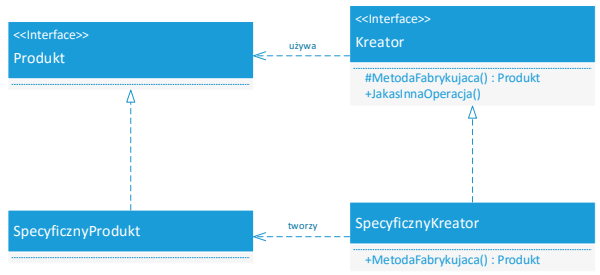


Metoda wytwórcza (fabrykująca)

PRZEZNACZENIE
Określa interfejs do tworzenia obiektów, umożliwiając jednak klasom przekazanie tworzenia egzemplarzy podklasom. Prościej mówiąc: jest to metoda, która może tworzyć obiekty o typie podanym przez tworzący je obiekt, a nie zdefiniowanym.

IMPLEMENTACJA
Implementacja wzorca powinna zawierać:
1. Interfejs lub klasę abstrakcyjną produktu.
2. Klasę lub klasy specyficznych produktów.
3. Interfejs lub klasę abstrakcyjną kreatora, zawierającą metodę wytwórczą.
4. Specyficzną klasę kreatora.

PRZYKŁADY ZASTOSOWANIA
- kiedy nie da się określić, jakie obiekty powinny być tworzone przez klasę bazową;
- kiedy implementacja jakiegoś narzędzia powinna być ukryta przed klientem, ale dawać mu możliwość modyfikacji jego działania;
- inicjalizacja sterowników;
- przy implementacji fabryki abstrakcyjnej.

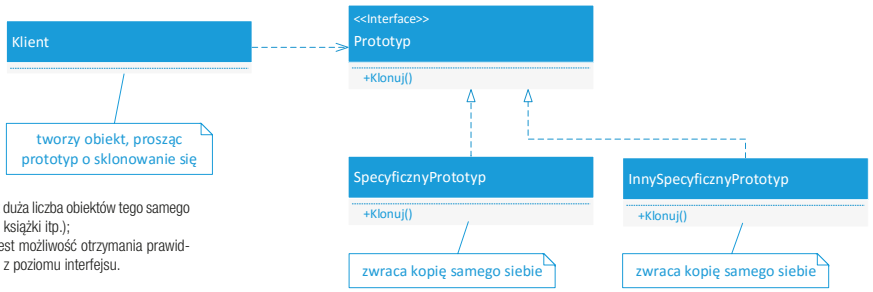


Prototyp

PRZEZNACZENIE
Umożliwia tworzenie nowych obiektów poprzez kopiowanie istniejącego z poziomu interfejsu.

IMPLEMENTACJA
1. Należy zadeklarować obiekt — prototyp, zawierający abstrakcyjną operację klonującą.
2. Klient może wywołać tę operację za każdym razem, kiedy chce stworzyć klon naszego obiektu.

ZASTOSOWANIE
- kiedy tworzona jest duża liczba obiektów tego samego typu (np. produkty, książki itp.);
- kiedy wymagana jest możliwość otrzymania prawidłowej kopii obiektu z poziomu interfejsu.

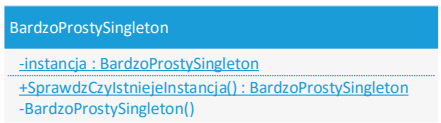


Singleton

PRZEZNACZENIE
Jego celem jest zapewnienie możliwości tworzenia obiektu o globalnym dostępie, który będzie miał pojedynczą instancję.

IMPLEMENTACJA
1. Tworzymy klasę, która posiada statyczną metodę, sprawdzającą, czy istnieje już jej instancja, a jeżeli nie, to ją tworzy i zapisuje jej referencję w prywatnym polu.
2. W klasie deklarujemy prywatny lub chroniony konstruktor, aby uniemożliwić dalsze tworzenie obiektów.

PRZYKŁADY ZASTOSOWANIA
- publiczny obiekt, np. do połączenia z bazą danych, który chcemy zainicjować na początku i nie tworzyć więcej jego instancji;
- konfiguracja aplikacji.



WZORCE STRUKTURALNE

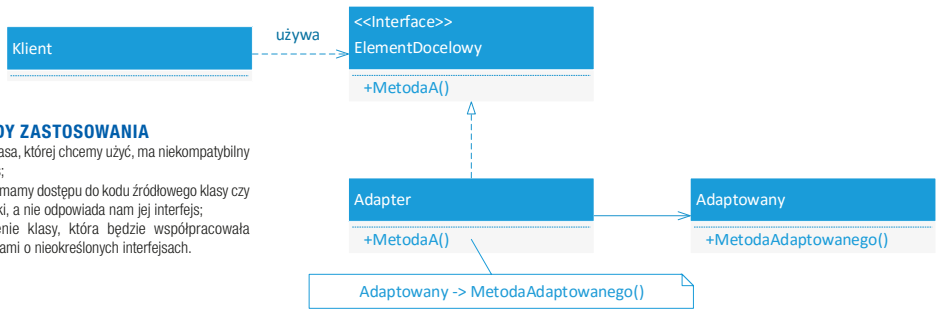
Opisują struktury powiązanych obiektów oraz sposoby składania obiektów w większe struktury.

Adapter (wrapper)

PRZEZNACZENIE
Jak sama nazwa wskazuje — przekształca coś, co jest dostępne, w coś oczekiwanego przez klienta. Jego zastosowanie możemy porównać do adaptera podróżnego, który zabieramy ze sobą, jadąc do USA czy Anglii.

IMPLEMENTACJA
1. Adapter implementuje abstrakcję, która jest opakowaniem dla obiektu.
2. Abstrakcja zapewnia przetłumaczenie protokołu klienta na protokół zrozumiały dla adaptowanego.

PRZYKŁADY ZASTOSOWANIA
- kiedy klasa, której chcemy użyć, ma niekompatybilny interfejs;
- gdy nie mamy dostępu do kodu źródłowego klasy czy biblioteki, a nie odpowiada nam jej interfejs;
- stworzenie klasy, która będzie współpracowała z obiektami o nieokreślonych interfejsach.

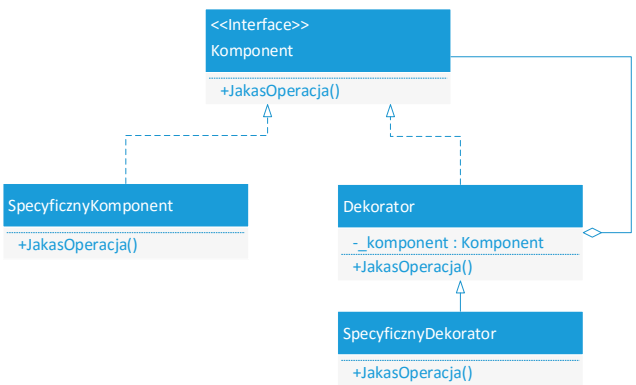


Dekorator

PRZEZNACZENIE
Umożliwia dynamiczne dodawanie nowych funkcji do istniejących klas. Można to nazwać dziedziczeniem w trakcie działania programu.

IMPLEMENTACJA
1. Dekorator tworzy klasę, która zawiera wszystkie metody klasy bazowej oraz dodatkowe funkcje. Zazwyczaj klasa bazowa jest przekazana jako parametr konstruktora w dekoratorze, a metody dekoratora poza dodaniem nowych funkcji wywołują te z oryginalnego obiektu.
2. Dekoratory mogą być łączone i wielokrotnie wykorzystywane.

PRZYKŁADY ZASTOSOWANIA
- dynamiczna zmiana wyglądu okna w zależności od potrzeb;
- dodawanie funkcji do istniejących klas w czasie działania programu.

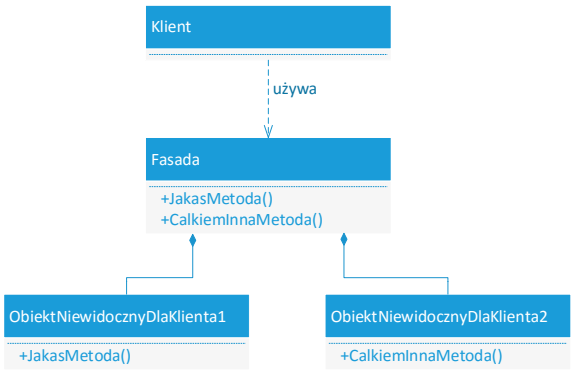


Fasada

PRZEZNACZENIE
Wzorzec ułatwia dostęp do złożonego systemu poprzez udostępnienie ujednolicionego interfejsu. W uproszczeniu można napisać, że fasada to obiekt udostępniający w jednym miejscu elementy z wielu obiektów systemu.

IMPLEMENTACJA
1. System złożony jest z wielu klas.
2. Klasa fasady posiada referencję do wszystkich klas, z których metody powinny być udostępnione klientowi.
3. Klient korzysta z fasady udostępniającej metody z poszczególnych klas.

PRZYKŁADY ZASTOSOWANIA
- ukrycie części systemu przed klientem (np. w banku, sklepie) i zmniejszenie liczby zależności pomiędzy klientem a systemem;
- API do połączenia z naszą aplikacją.

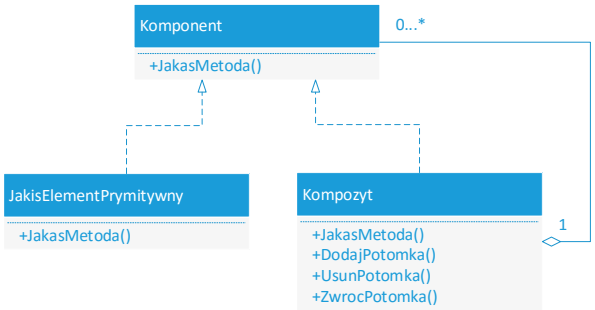


Kompozyt

PRZEZNACZENIE
Łączy obiekty w strukturę/hierarchię, aby można było z nich korzystać jednolicie dla kolekcji i pojedynczego obiektu.

IMPLEMENTACJA
1. W systemie, który składa się z wielu obiektów prymitywnych oraz kompozycji obiektów prymitywnych, tworzymy abstrakcję, która definiuje zachowanie wszystkich obiektów.
2. Wszystkie obiekty powinny implementować stworzoną przez nas abstrakcję.
3. Klient wykorzystuje kompozyt zamiast bezpośredniego korzystania z obiektów.

PRZYKŁADY ZASTOSOWANIA
- kiedy potrzebujemy systemu z możliwością łatwego rozszerzania o nowe komponenty;
- np. sprzedaż produktów, zestawów produktów, usług.

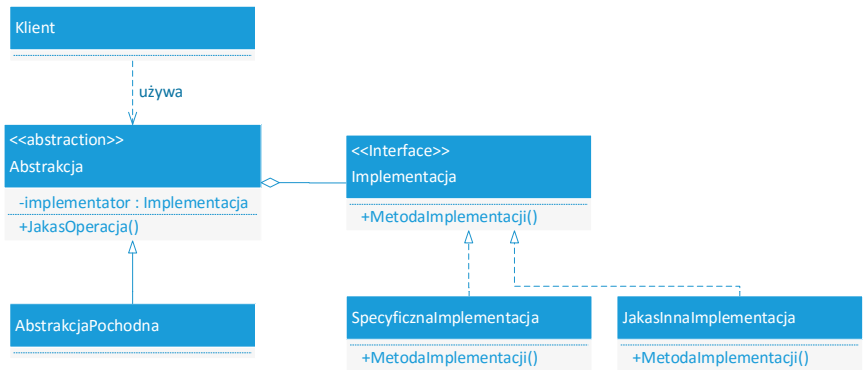


Most

PRZEZNACZENIE
Pozwala uniezależnić abstrakcję obiektu od jego faktycznej implementacji.

IMPLEMENTACJA
1. Zdefiniowany jest wspólny interfejs dla wszystkich konkretnych implementacji.
2. Interfejs wykorzystywany przez klasę abstrakcyjną, która przekazuje polecenia klienta do obiektu implementacji.

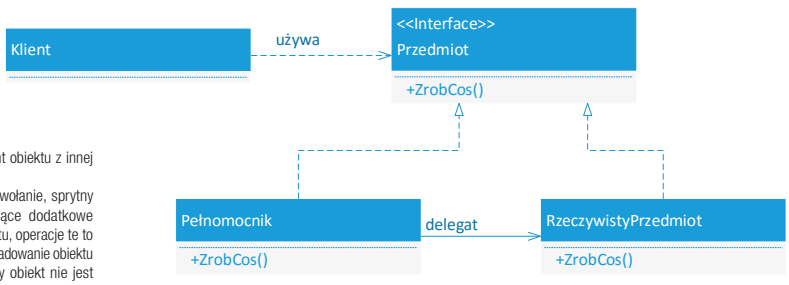
PRZYKŁADY ZASTOSOWANIA
- tworzenie obiektów przy użyciu różnych bibliotek (zarówno obiekty, jak i biblioteki mogą się zmieniać).



Pełnomocnik (proxy, substytut)

PRZEZNACZENIE
Tworzony jest obiekt będący odpowiednikiem i zastępujący inny obiekt. Umożliwiamy w ten sposób zarządzanie dostępem do tego obiektu.

IMPLEMENTACJA I ZASTOSOWANIE
Zasadę działania i zastosowania przy tym wzorcu należy uzależnić od typu pełnomocnika, którego użyjemy:
- wirtualny — przechowuje obiekty, których automatyczne tworzenie jest zbyt kosztowne (np. obraz), i tworzy je dopiero na żądanie;
- ochraniający — kontroluje dostęp do oryginalnego obiektu. Obiekty mogą mieć różne prawa dostępu — sprawdza on, czy obiekt wywołujący ma odpowiednie prawa do obiektu wywoływanego;
- zdalny (ambasador) — reprezentant obiektu z innej przestrzeni adresowej;
- inteligentne referencje (sprytne odwołanie, sprytny wskaźnik) — wskaźniki wykonujące dodatkowe operacje podczas dostępu do obiektu, operacje te to m.in. zliczanie referencji do obiektu, ładowanie obiektu do pamięci, sprawdzanie, czy dany obiekt nie jest zablokowany przed dostępem do niego.



Pyłek

PRZEZNACZENIE
Wzorzec stosowany do tworzenia dużej liczby identycznych (lub niewiele się od siebie różniących) obiektów, aby móc się nimi posługiwać w jednolity sposób. Wykorzystuje współdzielenie, przez co zmniejsza zapotrzebowanie aplikacji na pamięć, jednocześnie mogąc zmniejszyć wydajność.

IMPLEMENTACJA
1. Dane w obiekcie dzielone są na zewnętrzne i wewnętrzne:
   a. wewnętrzne (współdzielone) są modyfikowane jedynie przy inicjacji obiektu;
   b. zewnętrzne (unikatowe dla poszczególnych obiektów) mogą być dostarczone przed przekazaniem obiektu do klienta.
2. Fabryka pyłków zwraca klientowi istniejący już pyłek lub tworzy nowy, o ile wcześniej żaden egzemplarz nie został utworzony.

PRZYKŁADY ZASTOSOWANIA
- aplikacje złożone z bardzo wielu bardzo podobnych obiektów — edytory tekstu, edytory graficzne, programy architektoniczne.

WZORCE OPERACYJNE (CZYNNOŚCIOWE)

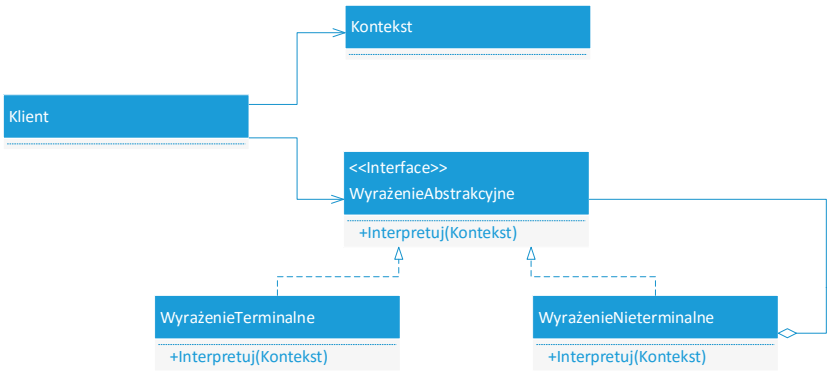
Opisują zachowanie obiektów, komunikację pomiędzy nimi i ich odpowiedzialność.

Interpreter

PRZEZNACZENIE
Jego zadaniem jest określenie opisu gramatyki języka oraz stworzenie interpretera, który będzie wykorzystywał ten opis do interpretowania zdań z tego języka.

IMPLEMENTACJA
1. Jeden z obiektów odpowiedzialny jest za przetrzymywanie danych do interpretacji.
2. Tworzymy abstrakcję, która interpretuje polecenia.
3. Dla poszczególnych przypadków tworzymy konkretne obiekty interpretujące dane i zgodne z naszą abstrakcją.
4. Abstrakcja wraz z obiektami konkretnych implementacji tworzy wzorzec metody szablonowej.

PRZYKŁADY ZASTOSOWANIA
- interpretacja języka.

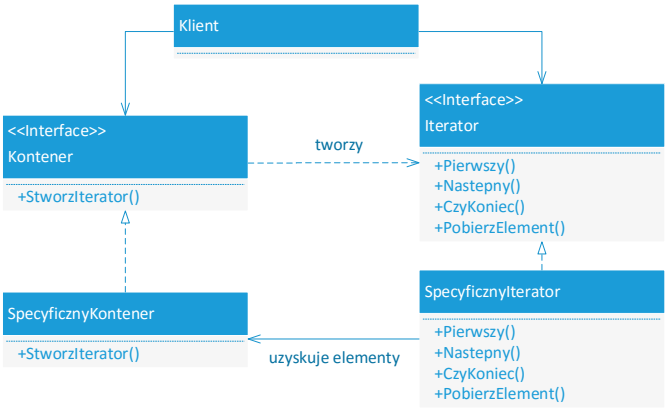


Iterator (kursor)

PRZEZNACZENIE
Umożliwia sekwencyjny dostęp do elementów złożonego obiektu bez ukazywania jego implementacji. W prostszych słowach: pomaga on w unifikowany sposób dostać się do elementów kolekcji, pomijając różnicę w ich implementacji.

IMPLEMENTACJA
Wzorzec podzielony jest na dwie grupy obiektów. Jedna z nich odpowiada za obsłużenie kolekcji, a druga za iterację po tych obiektach. Grupy te połączone są ze sobą interfejsami.

PRZYKŁADY ZASTOSOWANIA
- przetwarzanie zróżnicowanych kolekcji;
- aplikacje, w których dane są przechowywane w kolekcjach różnych typów.

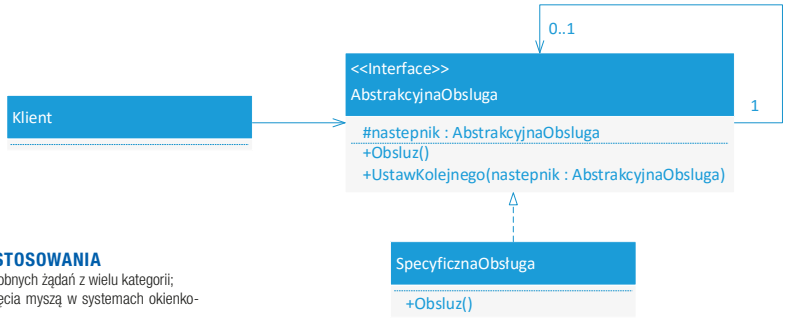


Łańcuch zobowiązań

PRZEZNACZENIE
Wzorzec odpowiedzialny za przekazanie odpowiedzialności za wykonywanie zadań do kolejnych obiektów, aż do jego wykonania. Łączy wiele obiektów w łańcuch i przekazuje żądanie między nimi. Jeśli obecny obiekt nie może wykonać zadania, przekazuje je do następnika. Jeśli ostatni obiekt nie może wykonać żądania, zostaje ono odrzucone. W życiu codziennym można to porównać do korporacji, w której odpowiedzialność jest przekazywana z rąk do rąk aż do momentu rozwiązania zadania.

IMPLEMENTACJA
Głównymi częściami wzorca są lista jednokierunkowa i abstrakcja, która jest szablonem dla wszystkich składników tejże listy.

PRZYKŁADY ZASTOSOWANIA
- mechanizm podobnych żądań z wielu kategorii;
- zdarzenie kliknięcia myszą w systemach okienkowych.

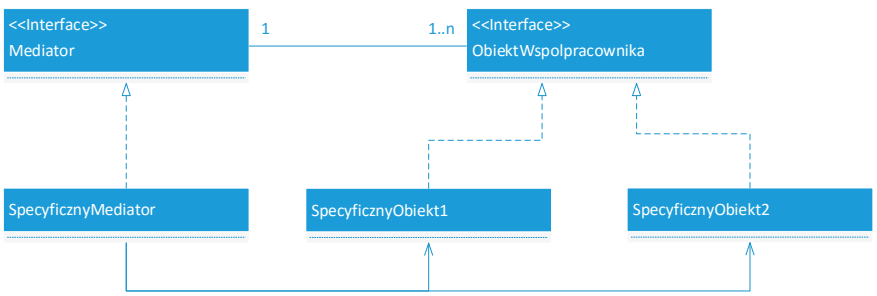


Mediator

PRZEZNACZENIE
Zapewnia zmniejszenie liczby powiązań pomiędzy poszczególnymi obiektami i jednolity dostęp do systemu dla klienta dzięki znajomości pozostałych obiektów systemu.

IMPLEMENTACJA
1. Klient tworzy kilka obiektów współpracujących ze sobą.
2. Mediator posiada wskaźniki na wszystkie obiekty, które mogą się ze sobą komunikować, i implementuje metody komunikacyjne.
3. Obiekty wywołują metody, które odnoszą się tak naprawdę do metod mediatora.

PRZYKŁADY ZASTOSOWANIA
- wiele obiektów o wspólnym interfejsie musi ze sobą współdziałać;
- np. przy obsłudze sklepu — producentach i konsumentach — jedni nie muszą wiedzieć o drugich, jedynie korzystają z produktów. Przy zastosowaniu wzorca konsumenci i producenci mogą być dowolnie zmieniani, dodawani, odejmowani.

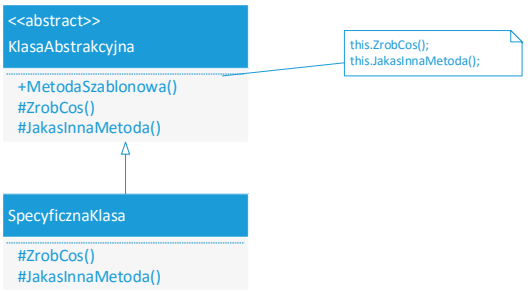


Metoda szablonowa

PRZEZNACZENIE
Określa szkielet algorytmu, jednocześnie pozostawiając możliwość jego modyfikacji i doprecyzowania w podklasach — bez zmiany struktury.

IMPLEMENTACJA
Klient korzystający z algorytmu może wykorzystać domyślną implementację lub utworzyć klasę pochodną i nadpisać metody w różniących się częściach algorytmu. Najczęściej metoda szablonowa jest publiczna, natomiast metody do przesłonięcia są chronione lub prywatne, aby klient nie mógł z nich bezpośrednio korzystać.

PRZYKŁADY ZASTOSOWANIA
- tworzenie podstawowego algorytmu, który może być nadpisywany, np. sortowanie.

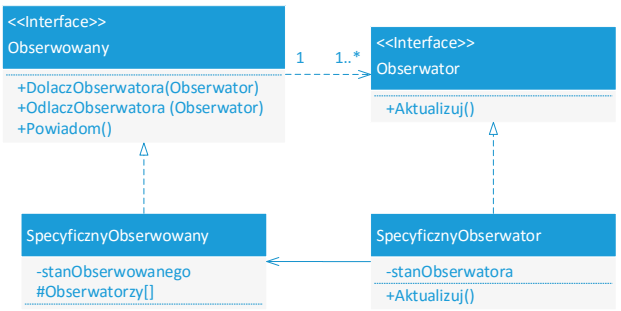


Obserwator

PRZEZNACZENIE
Jak sama nazwa wskazuje, wzorzec umożliwia obserwowanie obiektów i informowanie o ich zmianach inne zainteresowane obiekty.

IMPLEMENTACJA
Wzorzec określa zależności jeden do wielu pomiędzy obiektami. Obiekt obserwowany przekazuje listę zmian lub pozwala na wyciągnięcie tej listy poprzez przekazanie referencji do samego siebie. O zdarzeniu zmiany stanu obiektu informowane są wszystkie obiekty od niego zależne.

PRZYKŁADY ZASTOSOWANIA
- system informacyjny;
- prezentacja zależnych od siebie danych.

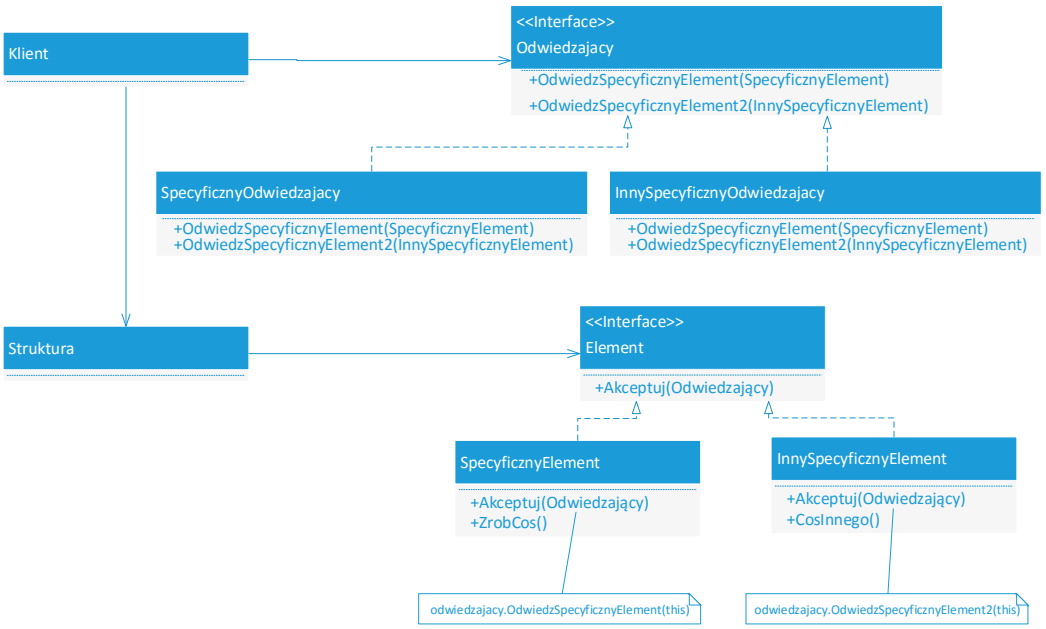


Odwiedzający (wizytator)

PRZEZNACZENIE
Zadaniem tego wzorca jest „odwiedzanie" obiektów i wykonywanie na nich operacji. Umożliwia on np. wielokrotne przejście po tej samej strukturze i za każdym razem zebranie innych danych — pozwala odseparować definicję algorytmu od struktury obiektów.

IMPLEMENTACJA
1. Należy stworzyć obiekt odwiedzającego, który zawiera deklarację wirtualnych metod odpowiadających za wizytację poszczególnych typów obiektów.
2. Wszystkie elementy struktury, które mogą być odwiedzone przez wizytatora, dziedziczą po klasie abstrakcyjnej (lub implementują interfejs) posiadającej metodę `accept()`, do której przekazywany jest konkretny odwiedzający.
3. Dla każdej operacji zdefiniowany jest obiekt konkretnego odwiedzającego, w którym definiuje się sposób działania metod odwiedzających.

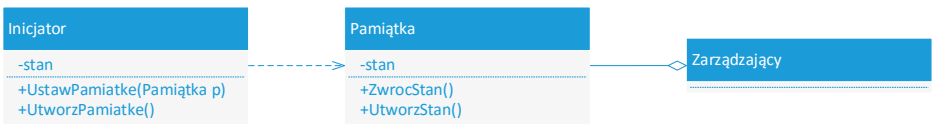
PRZYKŁADY ZASTOSOWANIA
- parsowanie XML-a;
- wszędzie, gdzie struktura pozostaje niezmienna, a do wykonania jest wiele operacji.



Pamiątka (znacznik)

PRZEZNACZENIE
Nie naruszając hermetyzacji (kapsułkowania), umożliwia zarejestrowanie stanu obiektu na zewnątrz, aby było możliwe przywrócenie go do zapamiętanego stanu.

IMPLEMENTACJA
1. Pamiątka przechowuje stan obiektu — może obejmować cały obiekt lub jego część. Jej dane dostępne są jedynie dla inicjatora, który ją utworzył.
2. Inicjator może stworzyć pamiątkę lub odtworzyć na jej podstawie stan.
3. Zarządzaniem kolejnością stanów zajmuje się obiekt zarządcy, który nie może manipulować zawartością pamiątek.



PRZYKŁADY ZASTOSOWANIA
- operacja *cofnij*;
- ponawianie operacji lub odtwarzanie informacji.

Polecenie

PRZEZNACZENIE

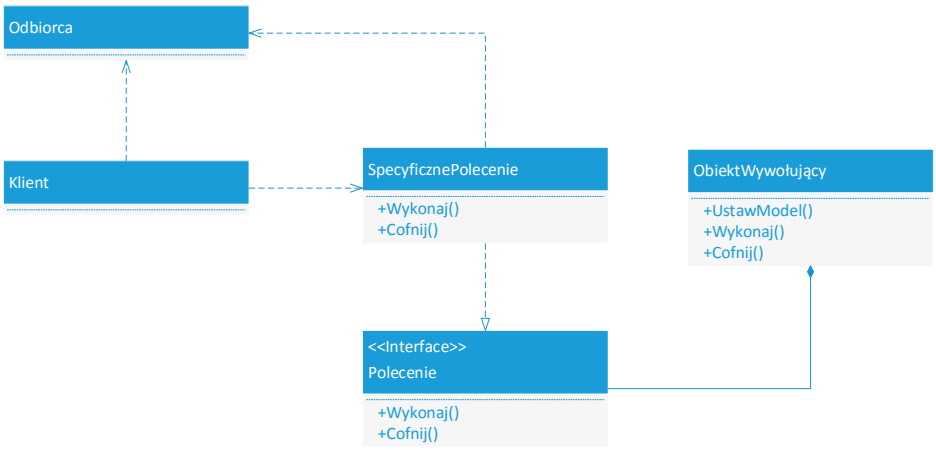
Odpowiada za enkapsulację żądań, co pozwala na ich parametryzację oraz kolejkowanie.

IMPLEMENTACJA

1. Definiujemy interfejs obiektu reprezentującego polecenie.
2. Każde polecenie implementuje konkretną akcję.
3. Definiujemy obiekt odpowiedzialny za wywoływanie poleceń — ustala on odbiorcę polecenia i wywołuje akcję przez niego zdefiniowaną.
4. Po wykonaniu polecenia efekt widoczny jest dla obiektu, który był odbiorcą.

PRZYKŁADY ZASTOSOWANIA

- menu w aplikacji.



Stan

PRZEZNACZENIE

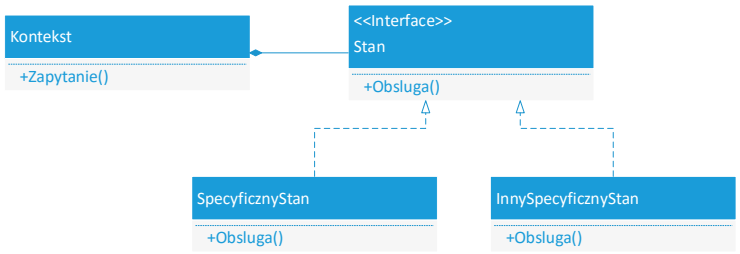
Umożliwia zmianę zachowania obiektu, kiedy zmieni się jego stan.

IMPLEMENTACJA

1. Wszystkie stany danego obiektu muszą mieć taki sam interfejs.
2. Stany są tymczasowe.
3. Zachowanie obiektu zależy od stanu, a zmiana stanu automatycznie powoduje zmianę w działaniu obiektu.

PRZYKŁADY ZASTOSOWANIA

- zastąpienie kaskadowego stosowania instrukcji warunkowej `if`;
- wszędzie, gdzie operacje uzależnione są od stanu — rachunek bankowy, konto klienta, połączenie z serwerem itp.



Strategia (polityka)

PRZEZNACZENIE

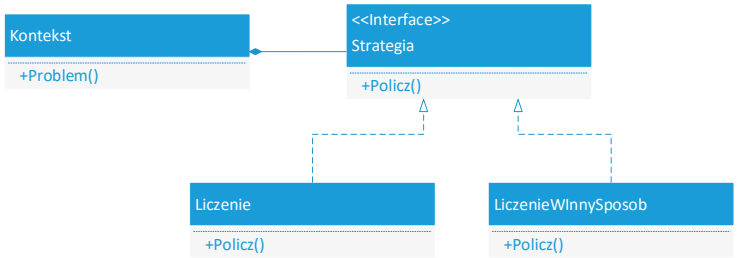
Umożliwia zamienne stosowanie różnych algorytmów poprzez zahermetyzowanie ich w postaci klas.

IMPLEMENTACJA

1. Tworzymy abstrakcję definiującą schemat dla rodziny algorytmów (może być interfejs lub klasa abstrakcyjna) — strategię.
2. Każda klasa konkretnych algorytmów dostosowuje się do ww. schematu, przez co można je dowolnie zamieniać.
3. Klient określa strategię i posiada do niej referencję.

PRZYKŁADY ZASTOSOWANIA

- wiele sposobów na wykonanie tej samej czynności;
- algorytmy, które są uzależnione od jakiegoś parametru (np. podatek VAT — zależy od regionu, w którym jest liczony).



ISBN: 978-83-283-4331-3

Wydawnictwo Helion
ul. Kościuszki 1c, 44-100 Gliwice
tel. +48 32 230 98 63
e-mail: helion@helion.pl
www.helion.pl