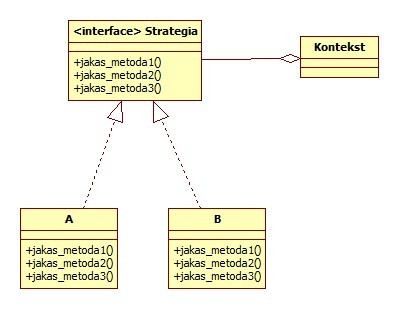
Wzorce projektowe opisują rozwiązania często powtarzających się problemów podając uniwersalne rozwiązania sprawdzone w praktyce. Rozróżniamy ich trzy główne rodzaje, są to: wzorce konstrukcyjne (creational), strukturalne (structural) oraz operacyjne/czynnościowe (behavioral). Pozwalają one skupić się na rozwiązywaniu rzeczywistego problemu. Są to szablony, a nie biblioteki gotowych do użycia klas czy metod. Algorytm może być jedynie implementacją wzorca projektowego, nie jest jednak tożsamy z tym pojęciem. Mogą one przyspieszyć proces rozwoju oprogramowania przez dostarczenie wypróbowanych rozwiązań dla problemów, które mogą nie być oczywiste na początku procesu projektowego. Często problemy te wiążą się z ewolucją oczekiwań względem projektowanego systemu: rozszerzeniem jego funkcjonalności, zmianą sposobu i formatu wprowadzanych danych czy dostosowaniem aplikacji do różnych klas użytkowników.

OOD Object-oriented design, metodyki projektowania, które wykorzystują pojęcia obiektowe, na które składają się: identyfikacja obiektów i ich klas; identyfikacja semantyki obiektów; identyfikacja związków pomiędzy obiektami; identyfikacja interfejsów obiektów i ich implementacji.

STRATEGIA

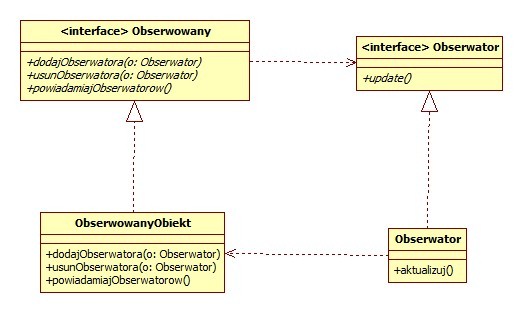
Istotą tego wzorca operacyjnego jest zapewnienie sposobu na elastyczne wybieranie algorytmu wykonania danej czynności w trakcie działania programu. Zastosowania mogą być przeróżne, jednym z częściej spotykanych w literaturze przykładów jest sklep internetowy, w którym to istnieje potrzeba naliczania podatku w zależności od kraju klienta. Do takiego problemu rozwiązaniem byłaby również prosta instrukcja warunkowa if lub switch, nie oferują one jednak takiej elastyczności jak wzorzec projektowy zarówno warunki wyboru jak i lista algorytmów byłyby trwale wpisane w program. Wzorzec definiuje rodzinę algorytmów, pakuje je jako osobne klasy i powoduje, że są one w pełni wymienne. Jego zastosowanie pozwala na to, aby zmiany w implementacji algorytmów przetwarzania były całkowicie niezależne od strony klienta, który z nich korzysta.



OBSERWATOR

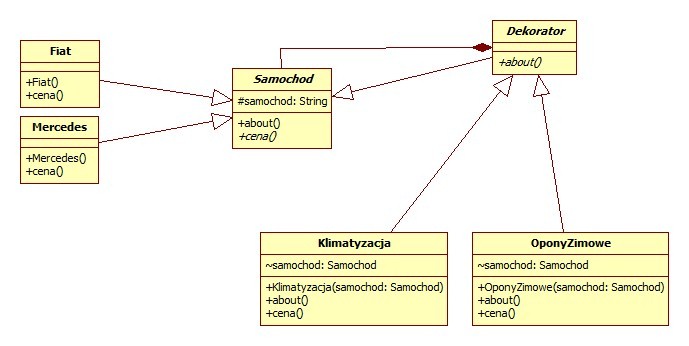
Obserwator to operacyjny wzorzec projektowy, umożliwiający stworzenie między obiektami zależności jeden-do-wielu. Zależność ta ma na celu automatyczne informowanie wielu obiektów o jakiejś zmianie stanu lub jakimś zdarzeniu i zostaną automatycznie zaktualizowane.

Gdy obserwatorzy dostają informacje, że dane uległy zmianie, pobierają je od obiektu obserwowanego i dokonują aktualizacji w posiadanych przez siebie danych, pod kątem stanu danych obiektu obserwowanego. Obiekt obserwujący sam może zdecydować, czy dalej chce obserwować, ale również obiekt obserwowany może go usunąć z listy obserwatorów. Obserwator nie zna innych obserwujących, są oni niezależni od siebie (jakakolwiek modyfikacja jednego obserwatora nie wpływają na innych).



DEKORATOR

Ten wzorzec strukturalny pozwala na dodawanie funkcjonalności do obiektu w czasie działania programu, stanowi więc alternatywę dla dziedziczenia, oferując rozszerzoną funkcjonalność. Dekorator pozwala na zmianę funkcjonalności, gdy instancja obiektu została już stworzona, dynamiczną zmianę przydzielonych mu zachowań.

Dekorator musi mieć taki sam interfejs jak obiekt, który będziemy dekorować (obiekty dekorujące są tego samego typu co obiekty dekorowane). Do dekoratora przekazujemy dekorowany obiekt. Klient wcale nie musi wiedzieć o jego działaniu. Zachowanie wzorca:

ADAPTER

Jadąc w podróż do Wielkiej Brytanii spodziewać się możemy, iż sprzęt elektryczny, który zabierzemy ze sobą nie będzie działał, nawet pomimo iż parametry angielskiej sieci wysokiego napięcia są podobne. Jednak wtyczka nie będzie pasowała do brytyjskiego gniazdka jako, że posiadają zupełnie różne standardy; różne interfejsy. Możemy jednak jeszcze przed wyjazdem kupić przejściówkę pozwalającą na podłączenie naszych urządzeń elektrycznych. Adapter jest właśnie taką przejściówką, wzorzec strukturalny pozwala na korzystanie z różnych systemów interfejsów. W programowaniu adapter będzie miał zastosowanie na przykład przy łączeniu projektowanego systemu z bazą danych. Jeżeli nasz system może być użyty w połączeniu z więcej niż jednym rodzajem bazy danych powinniśmy użyć klasy osłonowej w postaci Adaptera. Kod takiego programu napisany będzie w sposób uniwersalny, a w przypadku zmiany systemu bazy danych jedyną zmianą będzie podmiana sterownika w adapterze.

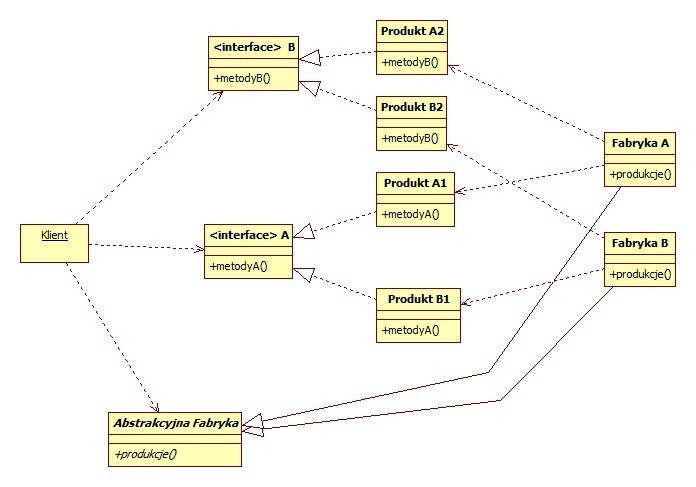
FACTORY

Wzorzec konstrukcyjny Factory pozwala na zwracanie przez metodę obiektu (lub przez funkcję) obiektów różnych klas w zależności od parametrów wejściowych. Pod pewnym względem można uznać go za podobny wzorzec do budowniczego, jednak w przeciwieństwie do niego wzorzec Factory zwraca interfejs konkretnej istniejącej klasy, a nie buduje interfejsu z różnych metod i właściwości, jak to ma miejsce w przypadku budowniczego.

Fabryka abstrakcyjna (*ang. abstract factory*) - pozwala tworzyć całe rodziny produktów. Mamy zdefiniowany interfejs do tworzenia spokrewnionych obiektów. Występuje tu kompozycja oraz pewne uzależnienie od abstrakcji, a nie klas rzeczywistych. Klient oddzielony jest od implementacji klas, tworzących obiekty.

Przykład:

Mamy fabrykę komputerów. Produkujemy laptopy i komputery stacjonarne PC. Wspomagamy się tutaj wzorcem zwanym metoda fabrykująca. Mamy tutaj główną klasę, klasę abstrakcyjną (po niej dziedziczą szczegółowe fabryki, np. na części do laptopów i części do komputerów PC). Szczegółowe fabryki produkują swoje podzespoły, tworzą nowe obiekty przy pomocy klas rzeczywistych. Klasy implementują odpowiadające im interfejsy. Teraz już tylko składamy komputery i wydajemy (metodo fabrykująca do dzieła!).



SINGLETON

Programowanie zorientowane obiektowo pozwala na tworzenie teoretycznie nieskończenie wielu instancji tego samego obiektu. Zdarzyć się jednak może, iż będziemy chcieli, aby każda nowa instancja obiektu odwoływała się do tych samych danych. Bardzo często dzieje się tak przy tworzeniu połączeń z bazą danych.

Tworzenie wielu połączeń nie przyspieszy operacji na bazie danych; tak czy inaczej zostaną one skolejkowane. Jedyną różnicą, zaś będzie dłuższy czas zużyty na stworzenie nowego połączenia do bazy danych.

Singleton, wzorzec konstrukcyjny, pozwala na odwoływanie się do tych samych właściwości (uchwytu do połączenia z bazą danych w naszym przypadku) dla każdego nowego obiektu danej klasy, który zostanie stworzony. W przypadku wykorzystywania wzorca Singleton używane jest też połączenie go z innym wzorcem, np. z Template.



FASADA

Wzorzec strukturalny, środek dostępu do złożonego systemu prezentujący na zewnątrz uproszczony lub uporządkowany interfejs programistyczny. Przykładem dla Fasady jest system obsługujący bankomat. Fasada występuje wtedy pomiędzy stosunkowo prostym bankomatem a rozbudowanym systemem bankowym, udostępniając jedynie funkcjonalność niezbędną do funkcjonowania bankomatu. Wpływa to na prostotę obsługi bankomatu oraz dodatkowo na bezpieczeństwo rozbudowanego systemu bankowego.

## Zastosowania:

* zwiększenie czytelności kodu poprzez wprowadzenie uproszczonego interfejsu do jego obsługi;
* zmniejszenie zależności pomiędzy plikami źródłowymi, co prowadzi do większej elastyczności w rozwoju projektu;
* uproszczenie zbioru źle zaprojektowanych interfejsów, jednym dobrze zaprojektowanym interfejsem.

ITERATOR

Iterator to obiekt pozwalający na sekwencyjny dostęp do wszystkich elementów lub części zawartych w innym obiekcie. Umożliwia on przetwarzanie każdego elementu w kolekcji bez konieczności znajomości jej wewnętrznej struktury. Obiekt może być więc traktowany przez użytkownika jako zwykła sekwencja lub lista. Wzorzec operacyjny ten stał się na tyle popularny, że nowoczesne języki programowania zawierają wbudowane funkcje do obsługi iteratorów.