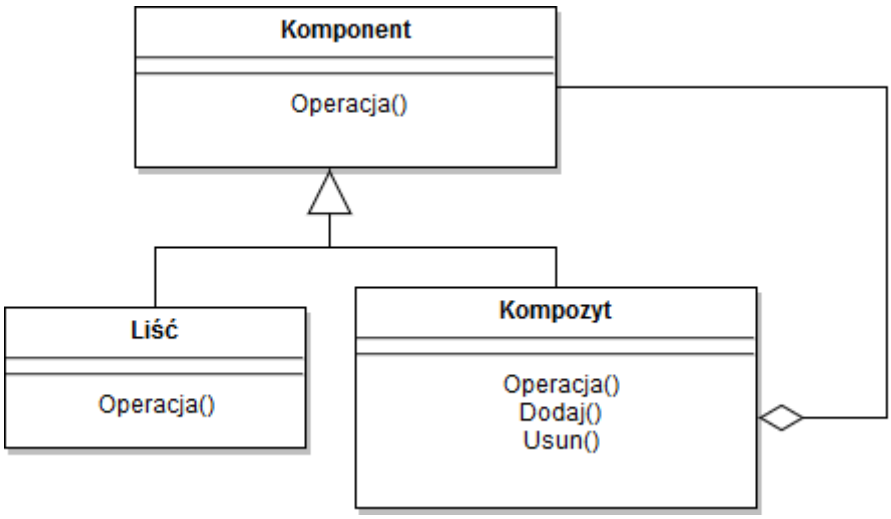
Wzorce projektowe

Kompozyt

Kompozyt to grupa obiektów, z których każdy może zawierać inne obiekty. Zatem każdy obiekt, może być grupą obiektów lub pojedynczym obiektem czyli liściem. Wzorzec kompozyt oparty jest o strukturę drzewiastą.



Kompozyt jak i Liść dziedziczy po tym samym interfejsie Komponent co pozwala na dostęp do obiektów tak samo jak do grupy tych obiektów. Możemy wykonywać operacje na pojedynczym obiekcie, jak i na grupie obiektów stosując ten wzorzec.

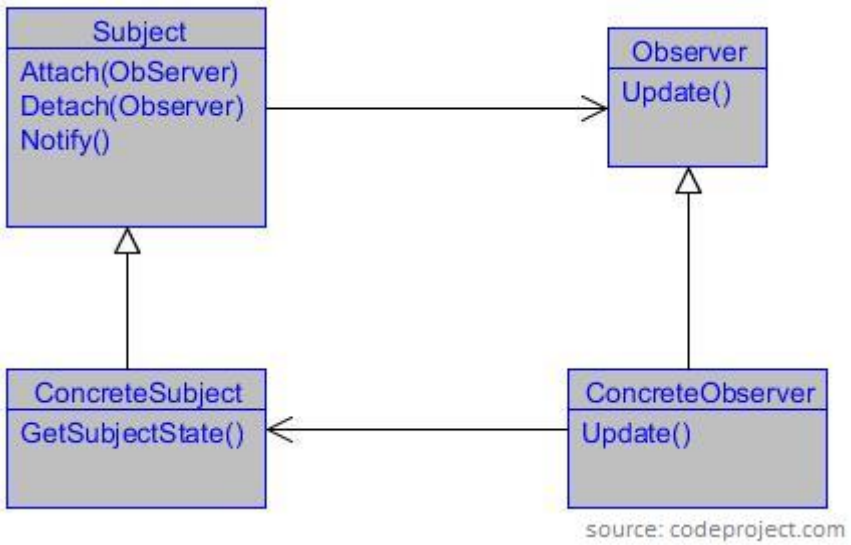
Wzorzec Kompozyt najlepiej sprawdzi się do zarządzenia grupą podobnych obiektów, na której możemy wykonać podobne operacje. Przez jedno wywołanie metody możemy sterować pojedynczym obiektem jak i całą grupą obiektów. Daje to wrażenie, że odwołujemy się do pojedynczego obiektu, gdy faktycznie działamy na grupie obiektów.

Obserwator

Obserwator to operacyjny (behawioralny) wzorzec projektowy, który umożliwia automatyczne powiadomienie i aktualizację obiektu (klienta) przez obiekt, który jest nasłuchiwany (tzw. subject).

Przede wszystkim sprawdza się on w przypadkach, kiedy musimy zaktualizować obiekt klienta, ale nie mamy jasnej informacji, kiedy subject zakończy swoją pracę.

W związku z tym ten wzorzec projektowy jest bardzo często wykorzystywany w aplikacjach wielowątkowych oraz systemach rozproszonych.



Subject - klasa ta zawiera listę wszystkich obserwatorów i dostarcza funkcjonalność pozwalającą nam dodawanie lub usuwanie obserwatora. Klasa ta jest również odpowiedzialna za aktualizacje obserwatorów, gdy dochodzi do jakiejś zmiany. W tym celu, w poniższym przykładzie została zaimplementowana klasa ASubject;

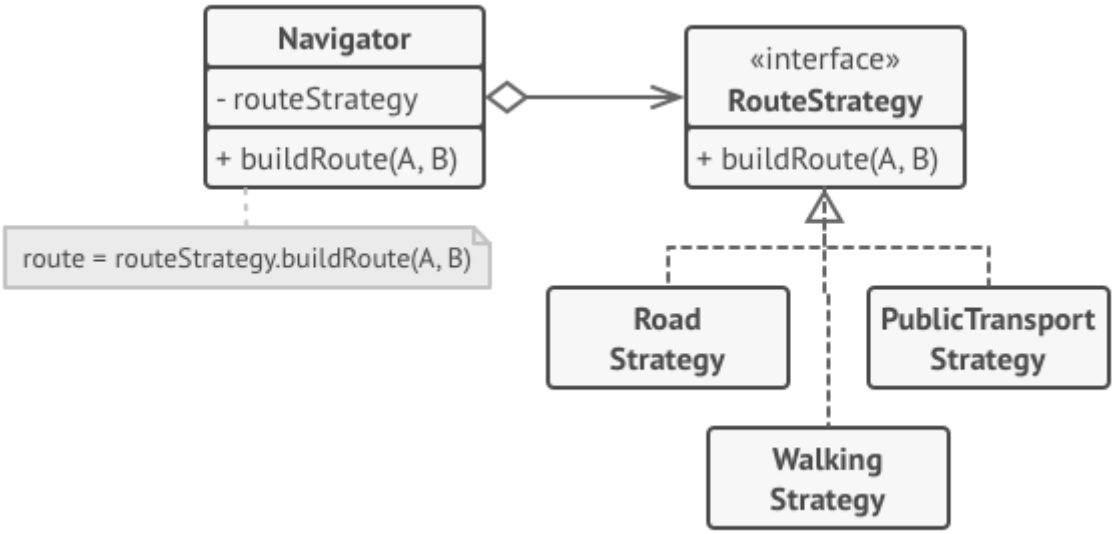
ConcreteSubject - klasa ta jest klasą implementującą Subject. Klasa ta jest encją, której zmiana wpłynie na wszystkie inne obiekty. W poniższym przykładzia została zaimplementowana klasa Dummy w celu osiągnięcia tego samego działania;

Observer - określa interfejs definiujący metody, które powinny być wywołane, gdy dochodzi do zmiany. W przykładowym projekcie jest to IObserver;

ConcreteObserver - to jest klasa, która musi aktualizować samą siebie wraz ze zmianą. Klasa ta musi zaimplementować Observer oraz zarejestrować siebie z ConcretSubject, aby otrzymywać powiadomienia.

Strategia

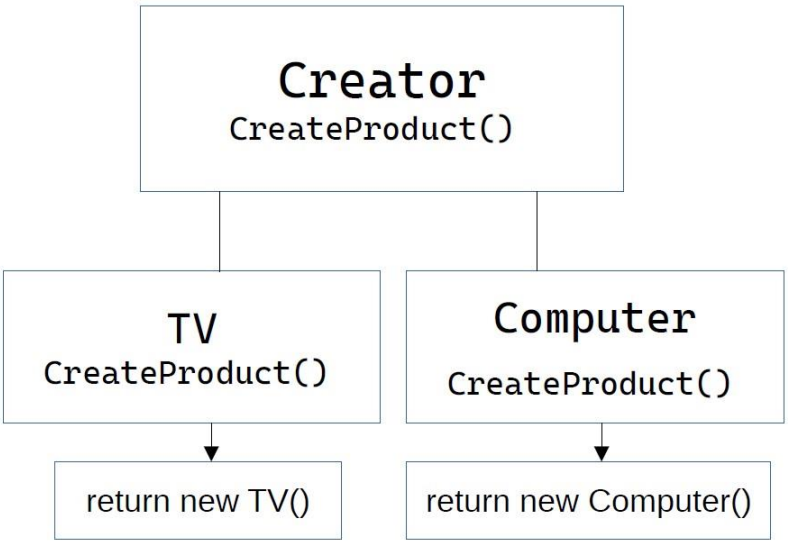
Tworząc oprogramowanie, musimy być przygotowani do obsłużenia sytuacji, w której kontekst zmienia się dynamicznie. W pewnych okolicznościach program może działać według jednego schematu, w innych zupełnie inaczej. Można by nawet powiedzieć, że ta sama aplikacja może mieć różne strategie działania. Takie warianty muszą być obsłużone niezależnie. Po uruchomieniu programu nie do końca wiadomo, który z nich zostanie użyty. Może to zależeć np. od parametrów wejściowych wprowadzonych na formatce czy też w linii komend. Aplikacja musi więc obsługiwać rodzinę różnych algorytmów, które mogą posłużyć do rozwiązywania problemów z określonej grupy. Takie algorytmy powinny działać wymiennie — tzn. że da się je wpiąć w to samo miejsce w kodzie i wywoływać odpowiednio w zależności od danych wejściowych.



Metoda Wytwórcza

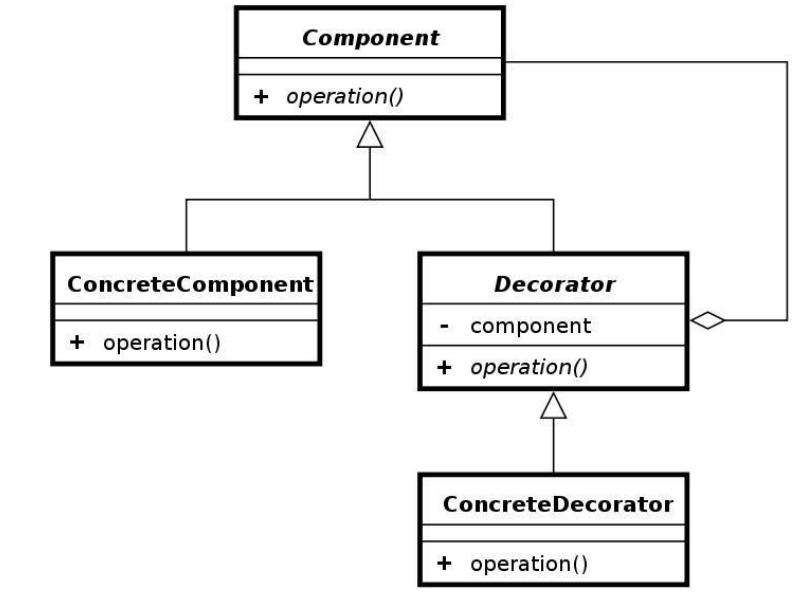
Metoda wytwórcza to wzorzec projektowy, którego celem jest dostarczenie interfejsu dla klas odpowiedzialnych za tworzenie konkretnego typu obiektów. Metoda wytwórcza definiuje standardowy sposób tworzenia obiektów w sposób niezależny od ich rodzaju. Rozwiązuje problem tworzenia obiektów bez określania ich konkretnych klas.

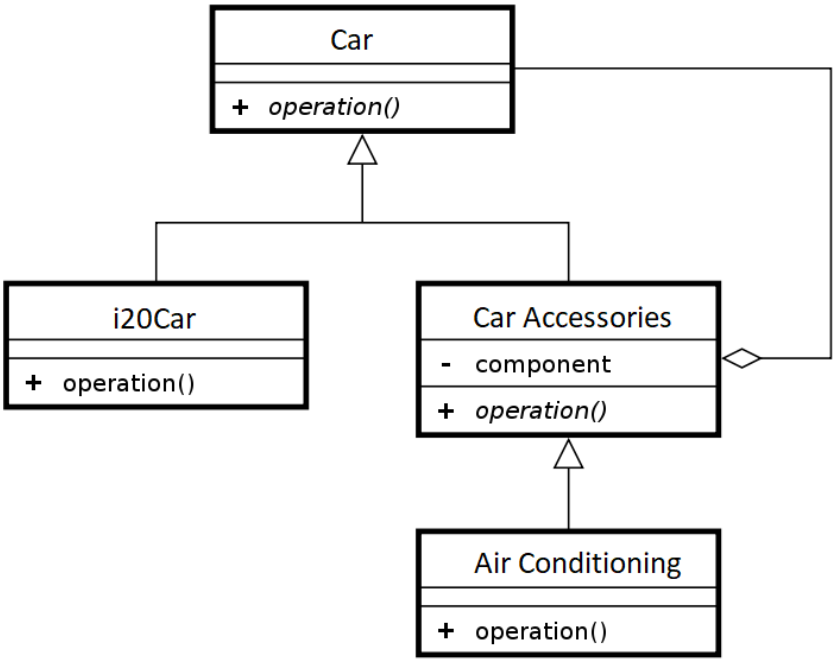
Metoda wytwórcza udostępnia interfejs do tworzenia obiektów w ramach klasy bazowej, ale pozwala podklasom zmieniać typ tworzonych obiektów. Ten wzorzec pozwala klasie odroczyć tworzenie instancji do podklas. Podklasy mogą nadpisać tę metodę w celu zmiany klasy tworzonych obiektów.



Dekorator

Dektorator to strukturalny wzorzec projektowy. Jest to jeden z najpraktyczniejszych wzorców projektowych, dlatego sprawdza się on bardzo dobrze w przypadkach gdzie mamy zaawansowaną hierarchię klas modelowych i zależności między nimi.





Zalety:

- Można rozszerzać zachowanie obiektu bez tworzenia podklasy.

- Można dodawać lub usuwać obowiązki obiektu w trakcie działania programu.

- Możliwe jest łączenie wielu zachowań poprzez nałożenie wielu dekoratorów na obiekt.

- Zasada pojedynczej odpowiedzialności. Można podzielić klasę monolityczną, która implementuje wiele wariantów zachowań, na mniejsze klasy.

Wady:

- Zabranie jednej konkretnej nakładki ze środka stosu nakładek jest trudne.

- Trudno jest zaimplementować dekorator w taki sposób, aby jego zachowanie nie zależało od kolejności ułożenia nakładek na stosie.

- Kod wstępnie konfigurujący warstwy może wyglądać brzydko.