## Kompozyt

**Kompozyt** to strukturalny wzorzec projektowy umożliwiający komponowanie struktury drzewiastej z obiektów i traktowanie jej jak pojedynczy obiekt.

**Przykłady użycia:** Wzorzec Kompozyt jest dość powszechny w kodzie C#. Często stosuje się go do modelowania hierarchii komponentów interfejsu użytkownika lub kodu który działa na grafach.

**Identyfikacja:** Jeśli klasy wszystkich obiektów w drzewie należą do jednej hierarchii to najprawdopodobniej mamy do czynienia z kompozytem. Jeśli dodatkowo metody tych klas delegują zadania obiektom-dzieciom wchodzącym w skład tego drzewa i robią to za pośrednictwem klasy bazowej lub bazowego interfejsu hierarchii, to na pewno jest to kompozyt.

**Przykład koncepcyjny**

Na postawie poniższych punktów może przedstawić strukturę wzorca **Kompozyt** :

* Z jakich składa się klas?
* Jakie role pełnią te klasy?
* W jaki sposób elementy wzorca są ze sobą powiązane?

## Obserwator

Obserwator jest stosowany w przypadku, kiedy pewne obiekty muszą być sprawnie poinformowane o zmianach występujących w innych obiektach,

szczególnie jeśli te zmiany występują często.

**Co umożliwia obserwator?**

Umożliwa elastyczne rozbudowywanie systemu

Umożliwa tworzenie mechanizmów słabo powiązanych

Udostępnia mechanizmy bezpośredniego powiadomiania o zmianach

**Gdzie jest używany obserwator?**

W Kontrolkach GUI

Aplikacjach wielowarstwowych czyli np GUI,logika

Aplikacjiach kontrolowanych zdarzeniowo

Systemach symulacyjnych

**Wytłumaczenie działania obserwatora.**

Obserwowany(Podmiot)

* Interfejs lub klasa abstrakcyjna definiująca operacje dodawania i usuwania obserwatorów

Konkretny obserwowany

* Klasa dziedzicząca po Obserwowany
* Zarządza swoim stanem
* W przypadku zmiany stanu obiektu powiadamia zarejestrowanych obserwatorów

Obserwujący

* Interfejs lub klasa abstrakcyjna definiująca operacje używane do powiadomienia obiektu tej klasy.

Konkretny obserwujący

* Konkretna implementacja Obserwującego
* W odpowiedni sposób reaguje na wywołanie operacji powiadamiającej

## Strategia

Wzorzec strategii może być traktowany jako wzorzec projektowy reprezentujący zachowanie, który jedocześnie jest prosty w implementacji i łatwy w zrozumieniu. Znajduje on zastosowanie w scenariuszach, w których obiekt będzie odpowiedzialny za wykonywanie wspólnych działań, które będą wybierane z szeregu dostępnych opcji.

Przejdźmy do zrozumienia definicji wzorca strategii, którą możemy odnaleźć w różnych zasobach sieci. "**Wzorzec strategii definiuje rodzinę algorytmów, które są odpowiedzialne za wykonanie zadania na kilka różnych sposobów. Algorytmy takie muszą zostać poddane hermetyzacji oraz mogą być zamieniane w trakcie wykonywania programu**”. Rozbijmy definicję na trzy części:

* **Rodzina algorytmów** - definicja ta mówi nam o tym, iż definiujemy kilka algorytmów, które mają tą samą funkcjonalność (np. sortowanie), ale proces ten wykonują w odmienny sposób;
* **Hermetyzacja** - odznacza, iż wzorzec wymusza od nas, aby algorytmy były umieszczone w różnych klasach. Takie zachowanie pomaga nam w wyborze odpowiedniego algorytmu dla naszego obiektu;
* **Zamienność** - zaleta wzorca polega na możliwości wyboru algorytmu w trakcie wykonywania programu oraz przypisania go do naszego obiektu.

## Metoda Wytwórcza

Ma na celu przeniesienie odpowiedzialności za wybór klasy tworzonego obiektu z klienta na klasę usługową

Wzorzec stosujemy gdy:

• Klasa nie może przewidzieć jakich typów obiekty musi utworzyć

• Klasa chce przenieść odpowiedzialność za wybór klasy tworzonych obiektów do swojej podklasy

**Opis w jaki sposób działa Metoda Wytwórcza**

## Produkt

•Definiuje interfejs obiektów tworzonych przez metodę fabrykującą Produkt Konkretny •Implementuje interfejs klasy Produkt

## Twórca

* Deklaruje metodę fabrykującą, która zwraca obiekt typu Produkt
* Może definiować domyślną implementację metody fabrykującej tworzącej domyślny Produkt Konkretny.
* Może inicjować Produkt(y) za pomocą metody fabrykującej

## Twórca Konkretny

* Nadpisuje metodę wytwórczą, żeby zwracała Produkt Konkretny

## Wady metody wytwórczej

* Klient może być zmuszony do tworzenia podklasy Twórcy

## Zalety metody wytwórczej

* Metoda fabrykująca eliminuje wstawianie do kodu konkretnych, specyficznych dla danej aplikacji klas.
* Kod operuje na zdefiniowanym interfejsie i abstrakcyjnym Twórcy, dzięki czemu kod może działać z dowolną klasą napisaną przez użytkownika

## Dekorator

**Dekorator** to strukturalny wzorzec pozwalający na dodawanie obiektom nowych obowiązków w sposób dynamiczny — poprzez “opakowywanie” ich w specjalne obiekty posiadające potrzebną funkcjonalność.

Stosując dekoratory można opakowywać obiekty wielokrotnie, gdyż zarówno obiekt docelowy jak i dekoratory są zgodne pod względem interfejsu. Wynikowy obiekt będzie posiadał ułożoną w formie stosu połączoną funkcjonalność wszystkich “opakowań”.

## Przykład koncepcyjny

Strukturę wzorca **Dekorator** można ze szczególnym naciskiem na pokazać przedstawiając następujące kwestie:

* Z jakich składa się klas?
* Jakie role pełnią te klasy?
* W jaki sposób elementy wzorca są ze sobą powiązane?