# WZORCE PROJEKTOWE – PODSTAWOWE DEFINICJE

Wzorzec projektowy - Zdefiniowany sposób rozwiązania określonego problemu w programowaniu. Wzorzec nie definiuje implementacji, a jedynie określa

sposób rozwiązania.

Wzorzec skłąda się z 4 kluczowych elementów:

1. Nazwy wzorca - sugerującej jego przydatnośc i do czego może byc zastosowany

2. Opisu problemu, do ktorego moze byc zastosowany wzorzec

3. Sposobu rozwiazania problemu

4. Konsekwencji i kosztow uzycia wzorca

Podział wzorców ze względu na zakres działania:

1. Wzorce konstrukcyjne - określają sposób budowania obiektów i klas

2. Wzorce strukturalne - określają sposób składania ze sobą klas i obiektów

3. Wzorce operacyjne - okreslają sposób współdziałania ze sobą obiektów i klas

Podział wzorców ze względu na zasięg:

1. Wzorce klas - dotyczą klas i interakcji miedzy klasami

2. Wzorce obiektów - dotycza obiektow i interakcji miedzy obiektami

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# WZORCE PROJEKTOWE - PODSTAWOWE DEFINICJE PROJEKTOWANIA OBIEKTOWEGO

Klasa - Czesciowa lub calkowita definicja dla obiektów. Klasa obejmuje zestaw możliwych interakcji (metody/ operacje) i zestaw danych

określających stan klasy (atrybuty)

Obiekt - jest instancją klasy, wirtualnym bytem posiadającym cechy zdefiniowane w klasie

Właściwości klas:

1. Enkapsulsacja - zdolnośc do ukrywania czesci informacji zwiazanych z klasa przed mozliwoscia dostepu z zewnatrz. Zmiana stanu zakapsulkowanej danej

wewnatrz klasy mozliwa jest tylko poprzez wywolanie operujacej na niej metody publicznej

2. Polimorfizm (wielopostaciowosc) - zdolnosc zastepowania danych obiektow innymi, z tym samym interfejsem, w czasie wykonywania programu.

Innymi slowy, jeden obiekt moze miec wiele postaci ale musi miec taki sam interfejs.

Rodzaje klas:

1. Abstrakcyjne - klasy, które definiują wyłącznie interfejs dla podklas, nie mają natomiast zaimplementowanego "jak działa"

2. Konkretne - klasy, które dostarczają konkretnego zestawu zachowan dla interfejsow zdefiniowanych

Mechanizm dziedziczenia - polega na przejmowaniu przez podklase interfejsu (dziedziczenie interfejsow) klasy nadrzednej.

* Dziedziczenie klas – polega na dziedziczeniu funkcjonalności obiektu (współużytkowanie kodu)
* Dziedziczenie interfejsów – polega na diedziczeniu samych interfejsów (wspiera polimorfizm)

W większości jezykow programowania dziedziczenie nie ma podzialu (jest 2 w 1), ale np. stworzenie abstrakcyjnej klasy interfejsowej (która z definicji nie realizuje funkcjonalności) prowadzi do dziedziczenia samego interfejsu.

**! Programuj pod kątem interfejsu, nie implementacji !**

**Otwarte powtórne wykorzystanie –** ponowne wykorzystanie funkcjonalności jednej klasy w innej kalsie, którego otwartość polega na jawności [klasa dziedzicząca przejmuje jawnie cały kod od klasy nadrzędnej] przejmowanego kodu (realizowane przez mechanizm dziedziczenia)

* Realizowane statycznie, na etapie kompilacji (nie można zmienić w traksie działania programu)
* Umożliwia powtóne implementowanie dziedziczonej metody (przysłanianie metod)
* WADA: klasy nadrzędne często określają część funkcjonalności klas podrzędnych. Zmiana więc funkcjinalności klasy nadrzędnej prowadzi do „zepsucia” klasy podrzędnej. Dziedziczenie interfejsów (po klasach abstrakcyjnych) nie ma tego problemu.

**Zamknięte powtórne wykorzystanie –** wykorzystanie funkcjonalności jednej klasy przez drugą, ale w sposób blackboxowy, realizowane poprzez składanie ze sobą obiktów (asocjacje, agregacje itp…)

* Jest realizowane dynamicznie, poprzez przypisywanie referencji między obiektami
* Wymaga pasujących interfejsów między obiektami

**! Przedkładaj kompozycję obiektów nad dziedziczenie klas !**

**Typy sparametryzowane <template> -** Pozwala zdefiniować typ bez konieczności definiowania wszystkich typów składowych. Typ składowy jest podawany w momencie tworzenia obiektu jako parametr. To pozwala stworzyć różne obiekty sparametryzowanego typu.

* Jest realizowane statycznie, na etapie kompilacji. Po utworzeniu obiektu danego, sparametryzowanego typu nie ma możliwości zmiany parametru.
* Wydajne, pozwala na zmiany typów używanych w klasie, co w efekcie może prowadzić do uproszczenia realizacji danej funkcjonalności

**Delegowanie –** podejście polegające na tym, że do obsługi danego żądania służą 2 obiekty – delegujący i delegowany. Obiekt delegujący deleguje obiektowy delegowanemu dane żądanie. Przypomina to dziedziczenie, z tą różnicą, że w dziedziczeniu obiekt podrzędny ma podziedziczony zestaw metod do realizacji danego żądania, a w delegowaniu to drugi obiekt (delegowany) realizuje dane żądanie przy użyciu własnych metod. W tym podejściu mamy do czynienia nie z dziedziczeniem, a z kompozycją obiektów, co pozwala na dynamiczne podmienianie współpracujących obiektów, ale też podnosi poziom skomplikowania kodu (realizacja funkcjonalności staje się dynamiczna, a nie statyczna, jak w przypadku dziedziczenia). Delegat może również delegować sam siebie do obsługi danego żądania (realizujemy w ten sposób zdolność przy dziedziczeniu do obsługi funkcjonalności przez samego siebie).

Interfejsy obiektow:

Kazda metoda ma swoja nazwe, rodzaj zwracanych i przyjmowanych argumentow. Ten zbior to Sygnatura metody, a zbior wszystkich sygnatur wszystkich

metod obiektu to interfejs obiektu.

Typy obiektow:

Mowimy, ze obiekt jest danego typu jesli przyjmuje wszystkie żadania interfejsu danego typu. Co wiecej, interfejs obiektu moze byc szerszy niz interfejs

danego typu.

Podtyp:

Mowiy, ze obiekt jest podtypem, jesli spelnia interfejs typu nadrzednego (dziedziczy po nim)

Zastosowanie interfejsow pozwala uzywac tzw. wiazan dynamicznych, czyli obiekty traktowane sa jak czarne skrzynki, wymiene, ale spojne co do interfejsow

(polimorficzne).

POŁĄCZENIE MIĘDZY OBIEKTAMI:

**Agregowanie –** gdy jeden obiekt ma drugi obiekt, tzn. jeden obiekt ma drugi lub obiekt agregowany jest częścią obiektu agregującego. Implementacyjnie (w cpp) – atrybutem jednego obiektu jest drugi obiekt (choć może też być referencja do innego obiektu :o to też jest agregacja (zależnie od tego jak to działa), ale z poziomu kodu niezauważalna)

**Asocjacja –** luźne powiązanie, oznacza, że jeden obiekt wie o istnieniu drugiego. Implementowane przez referencję (wskaźnik) do obiektu asocjowanego.

WZORCE KONSTRUKCYJNE:

1) Adapter – umożliwia dostosowanie interfejsu jednej klasy do interfejsu drugiej. Adapter to klasa współpracująca z 2 kalsami: targetem i adaptee. Adapter pozyskuje interfejs targetu poprzez dziedziczenia, natomiast przesłania podziedziczone metody w taki sposób, żeby wywoływać określone metody obiektu adaptowanego w celu wywołania określonej funkcjonalności. Adapter dziedziczy po targecie i ma referencję (wkaźnik; skłądanie obiektów) do adaptee (adaptet obiektowy); lub dziedziczy po obu klasach (adapter klasowy).

2) Dekorator – w dynamiczny sposób umożliwia dodanie nwych funkcjonalności do obiektu, niejako opakowuje obiekt nową klasą z rozszerzoną funkcjonalnością. Dekorator ma interfejs zgodny z klasą dekorowaną (dziedziczą po jednej abstrakcji). Iliość dekoratoró może być dowolna, ponieważ dla klientów dekorator jest „przeźroczysty”, tzn niezauważalny. Jeżeli chcemy uzywać więcej niż jednego dekoratora, robimy klasę abstrakcyjną po któej dziedziczą wszystkie dekoratory konkretne. Dekorator wykonuje operacje na obiekcie, ma referencję do dekorowanego obiektu (skłądanie obiektów). Metody dekoratora wywołują metody dekorowanego biektu, ale mogą też dodawać nowe funkcjinalności w swej implementacji.

3) Fasada – Umożliwia wystawienie jednego, uproszczonego, wspólnego interfejsu dla zestawu klas. Stosowana, gdy chcemy wystawć wspólny, uproszczony interfejs lub gdy chcemy ukryć interfejsy klas konkretnych (z których skłąda się większy system). Fasada uniezależnia klienty od systemu, tworzy niejako „panel” dla większego, złożonego systemu, ułatwia korzystanie z podsystemów. Fasada zawiera obiekty podsystemu, a metody fasady (interfejs) wywołują określone metody obiektów podsystemu.

4) Kompozyt – Umożliwia uporządkowanie obiektów w złozóne, drzewiaste struktury. We wzorcu kompozyt mamy 2 zasadnicze elementy: liść – podstawowy obiekt, niedekomponowalny i kompozyt – obiekt skłądający się zkompozytó lub liści. Cechą kompozytu jest to, że zarówno liście jak i kompozyty mają spójny, wspólny interfejs, toteż klienty mogą korzystać zarówno z kompozytów jak i z liści zamiennie. Kompozyty i liście dziedziczą zatem po wspólnej klasie abstrakcyjnej narzucającej interfejs. Kompozyty mają powiązanie z liścmi poprzez skłądanie obiektów (referencje) do podobiektów.

5) Most – umożliwia oddzielenie abstrakcji od implementacji, dzięki czemu można niezaleznie modyfikować te 2 obiekty. Wzorzec most realizuje to co robi dziedzicenie, ale przez skłądanie obiektów, więc dynamicznie. Wzorzec most wykorzystuje 2 obiekty: abstrakcję, któa definiuje interfejs i ma referencję do implementacji. Abstrakcja, poprzez referencję, wywołuje metody z implementacji. Jeżeli mamy więcej niż jedną implementację, implementacje dziedziczą po wspólnje klasie interfejsowej (muszę mieć zgodny interfejs żeby abstrakcje mogły z nich koryzstać jednakowo)

6) Pełnomocnik (Proxy) – jest „zastępcą” obiektu. Pełnomocnika wykorzystujemy, gdy z jakiś powodó nie chcemy używać oryginalnego obiektu (np. z powodu kontroli dostępu lub oszczędności pamięci). Pełnomocnik ma interfejs zgodny w obiektem, któy reprezentuje, ale nakłąda dodatkowe zachowanie na obiekt. Rodzaje pełnomocnikó: zdalny(pełnomocnk obiektu z innej przestrzeni adresowej), wirtualny (worzy obiekt na żądanie), zabezpieczający (kontroluje dostęp), inteligentny wskaźnik (robudowany wskaźnik). Pełnomocnik ma referenję do obiektu który reprezentuje i wywouje na nim określone metody.

7) Pyłek – obiekt, który może być współdzielony. Pyłki są specjalnymi rodzajami obiektów, które mogą yć współdzielone (reużywalne) w ramach jednego systemu. Używane są, gdy chcemu oszczędzić miejsca (a musimy użyać dużej liczby obiektów). Pyłki mają 2 zasadnicze cechy: stan zewnętrzny i wewnętrzny. Stan zewnętrzny jest zapisany w pyku i jest niezmienny. Stan wewnętrzny może być zmieniany przez kienty, przez co pyłek jest prametryzowany i dostosowany do aktualnej potrzeby użycia, co umożliwia reużywalność takiego obiektu. Przykłądem pyłków są znaki ASCII w edytorze tekstowym. Stan zewnętrzny to rodzaj znaku, natomiast wewnętrzny to czcionka i pozycja – reużywalny. Używanie pyłkó zwiększa narzut na obliczneia (trzeba obliczyć stan wewnętrzny za każdym razem gdy chcey używać pyłków), ale znacznie ogranicza koszt pamięciowy dla systemu.