# 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.