



akademia kodu

2019

Wzorce projektowe

Historia

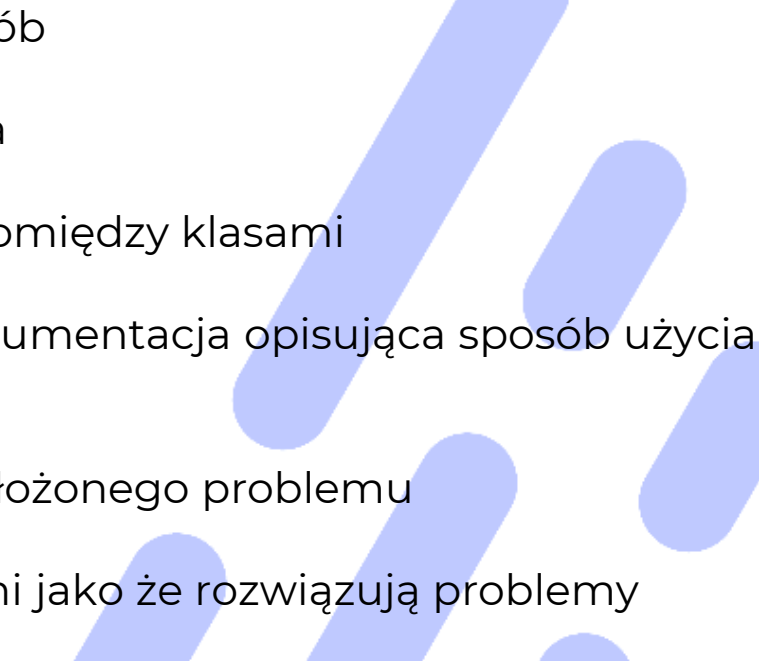
Wzorce projektowe w informatyce wywodzą się z wzorców projektowych w architekturze, które zostały zaproponowane przez austriackiego architekta Christophera Alexandra i miały ułatwić konstruowanie mieszkań i pomieszczeń biurowych. Pomysł ten nie został jednak przyjęty.

Inaczej stało się w informatyce. Termin wzorca projektowego został wprowadzony do inżynierii oprogramowania przez Kenta Becka oraz Warda Cunninghama w 1987 roku. Na konferencji OOPSLA, przedstawili oni wyniki swojego eksperymentu dotyczącego ich zastosowania w programowaniu. Został spopularyzowany w 1995 przez Gang of Four (Erich Gamma, Richard Helm, Ralph Johnson oraz John Vlissides) dzięki książce ***Inżynieria oprogramowania: Wzorce projektowe.***

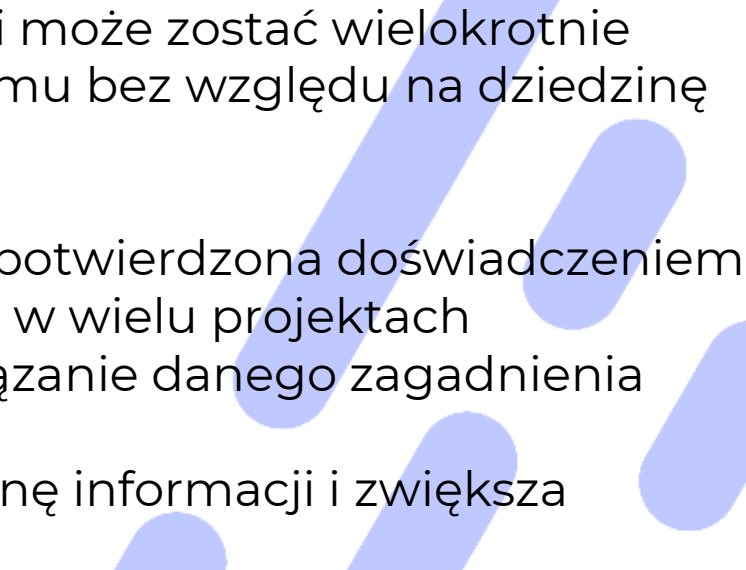
Co to wzorzec projektowy?

Wzorzec projektowy (ang. design pattern)- standardowe rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. Pokazuje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie, modyfikację oraz pielęgnację kodu źródłowego. Jest opisem rozwiązania, a nie jego implementacją. Wzorce projektowe stosowane są w projektach wykorzystujących programowanie obiektowe.

Czym są wzorce projektowe?

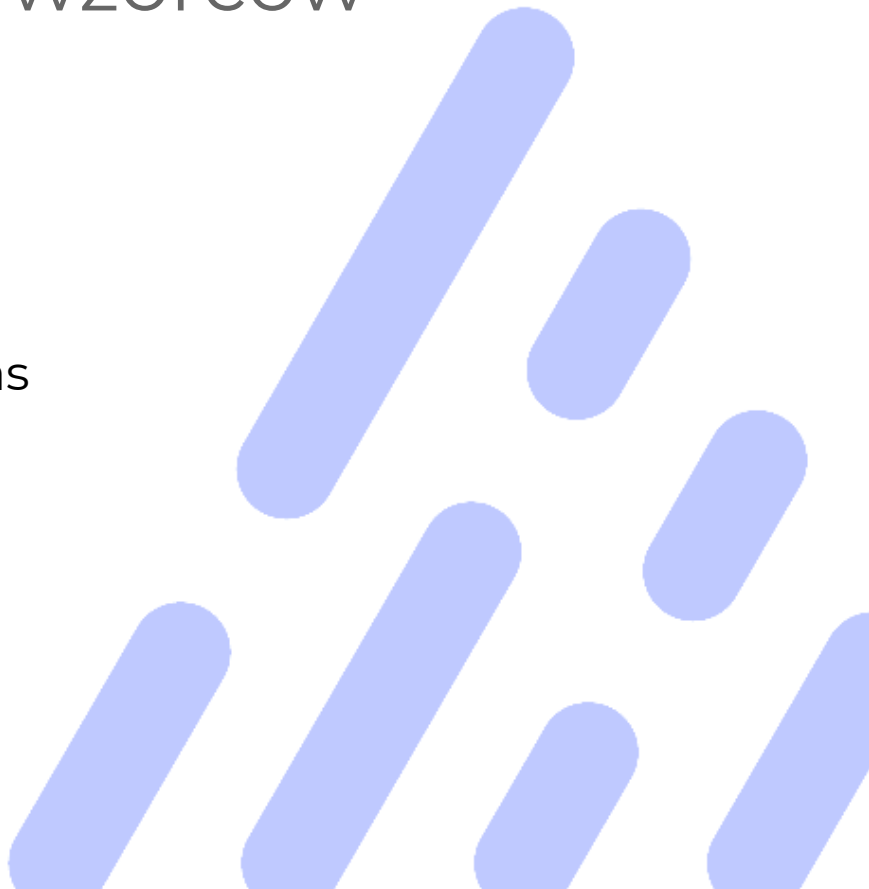
- ✓ Rozwiązanie problemu w określony sposób
 - ✓ Opisuje problem który się stale powtarza
 - ✓ Stanowią abstrakcyjny opis zależności pomiędzy klasami
 - ✓ Znaczna ich część stanowi obszerna dokumentacja opisująca sposób użycia wzorca
 - ✓ Są łączone w celu rozwiązania bardziej złożonego problemu
 - ✓ Algorytmy nie są wzorcami projektowymi jako że rozwiązują problemy obliczeniowe, a nie projektowe
- 

Jaki jest wzorzec projektowy?

- ✓ Reużywalny – w takiej samej postaci może zostać wielokrotnie stosowany do rozwiązywania problemu bez względu na dziedzinę projektu/oprogramowania
 - ✓ Sprawdzony – przydatność wzorca potwierdzona doświadczeniem nabytym poprzez jego zastosowanie w wielu projektach informatycznych – optymalne rozwiązanie danego zagadnienia
 - ✓ Powszechnie znany – ułatwia wymianę informacji i zwiększa czytelność kodu
- 
- The background of the slide features several thick, light blue diagonal bars of varying lengths and orientations, creating a modern, abstract design.

Ogólny podział wzorców

- ✓ **klasowe** - dotyczące klas;
- ✓ **obiektowe** - dotyczące obiektów klas



Szczegółowy podział wzorców projektowych

✓ **Wzorce kreacyjne:**

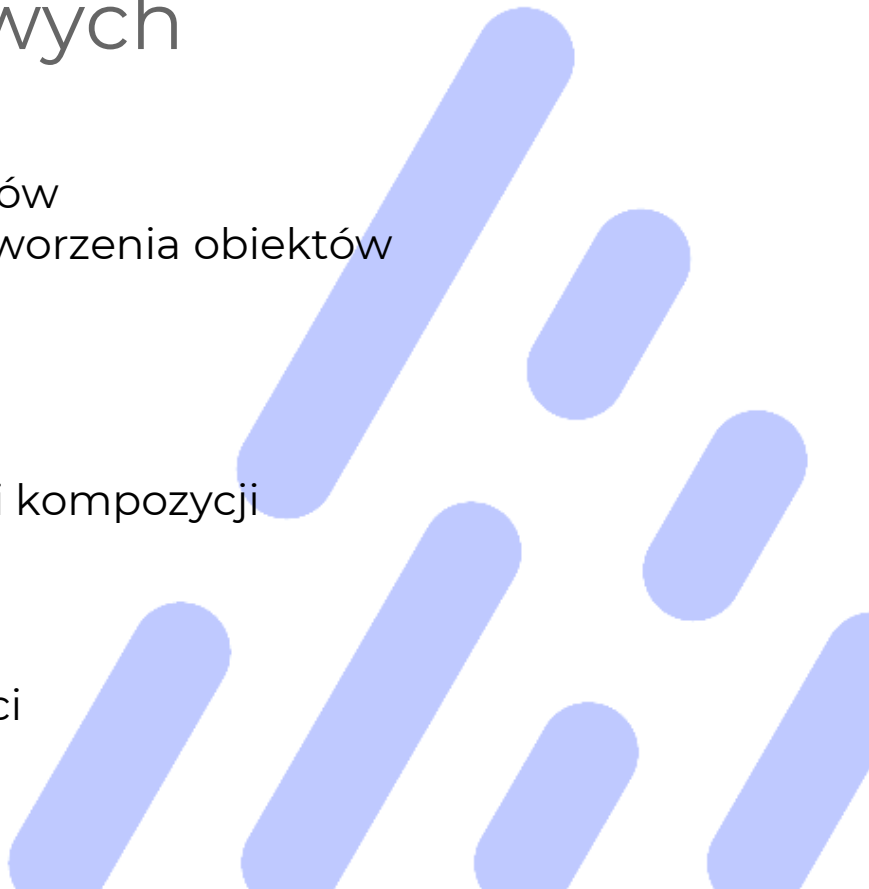
- abstrakcyjne metody tworzenia obiektów
- uniezależnienie systemu od sposobu tworzenia obiektów

✓ **Wzorce strukturalne:**

- sposób wiązania obiektów w struktury
- właściwe wykorzystanie dziedziczenia i kompozycji

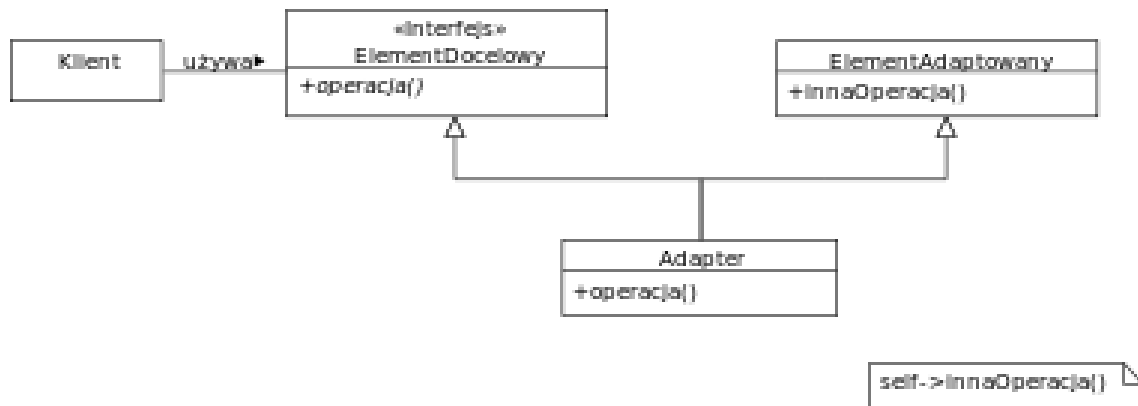
✓ **Wzorce czynnościowe:**

- algorytmy i przydział odpowiedzialności
- opis przepływu kontroli i interakcji



Wzorzec- Adapter (Wrapper)

Adapter dostarcza opakowanie wraz z pożądanym interfejsem, dzięki czemu pozwala na dopasowanie użycia istniejących obiektów do nowych klas



Wzorzec- Fasada (Facade)

Wzorzec fasady polega na tym, że tworzymy klasę, której jedynym zadaniem jest wywoływanie odpowiednich metod z innych klas (np. serwisów) czasem w odpowiedniej kolejności lub dodając/modyfikując pewne informacje.

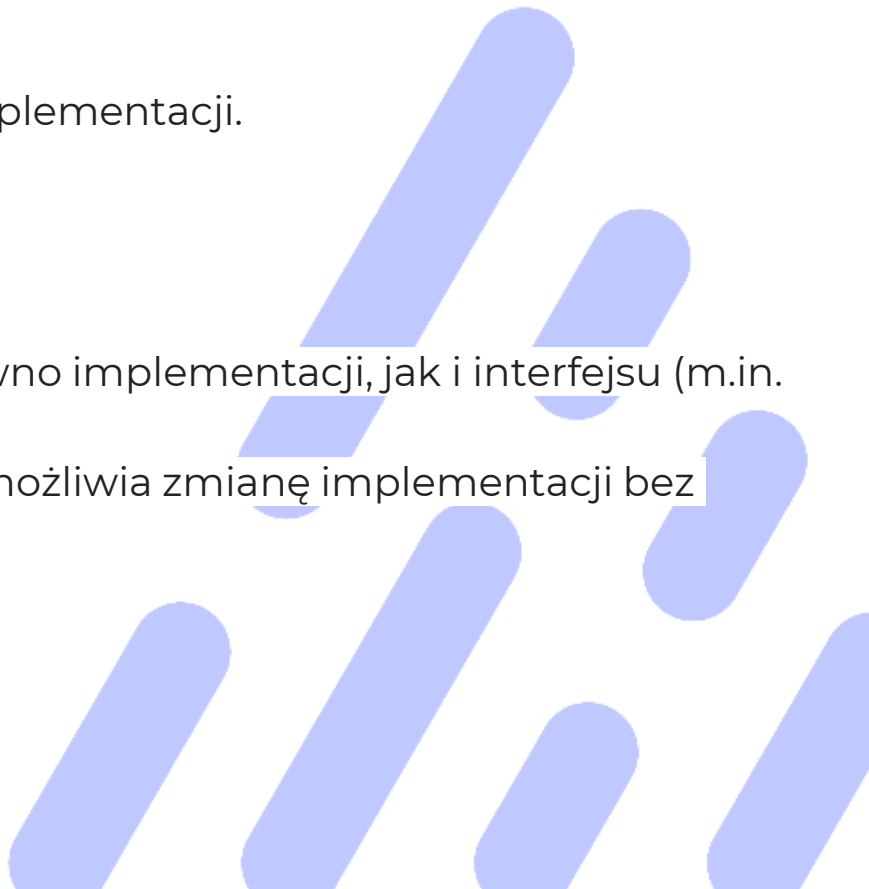
W praktyce fasady często spotykamy w sytuacjach, kiedy mamy wiele różnych systemów (np. w korporacjach), a potrzebujemy spójnego sposobu na dostęp do danych z różnych źródeł (np. dla działu marketingu potrzebujemy danych ze sprzedaży oraz z magazynu). Fasada ułatwia dostęp do różnych obiektów i ukrywa szczegóły implementacji.

Wzorzec -Most (Bridge)

Pozwala oddzielić abstrakcję obiektu od jego implementacji.

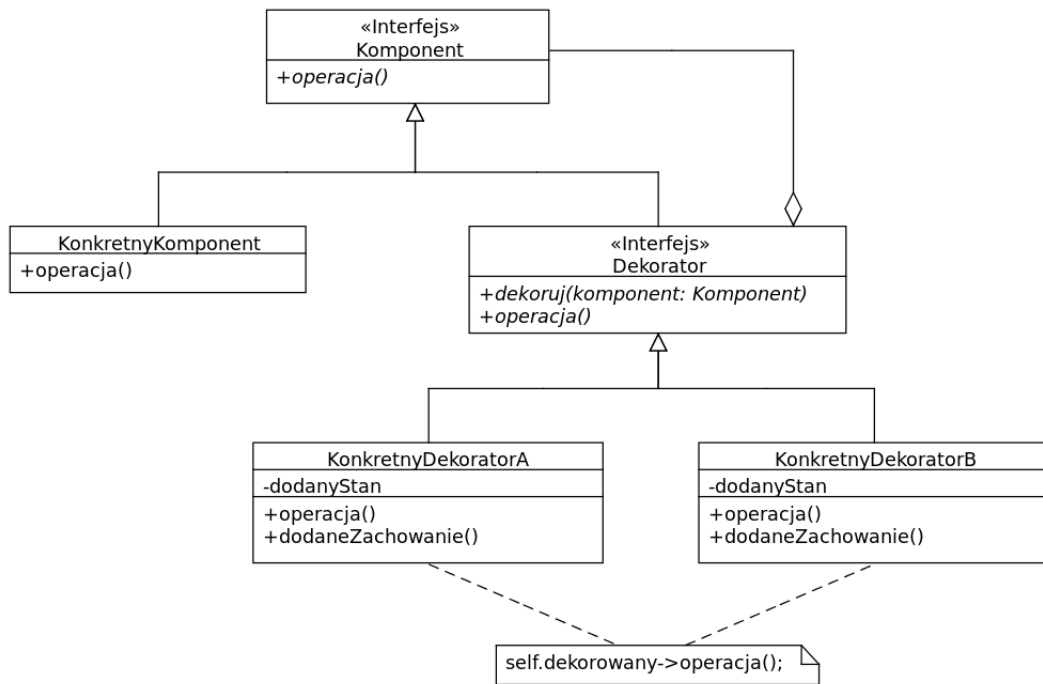
Zaleca się stosowanie tego wzorca aby:

- ✓ odseparować implementację od interfejsu,
- ✓ poprawić możliwości rozbudowy klas, zarówno implementacji, jak i interfejsu (m.in. przez dziedziczenie),
- ✓ ukryć implementację przed klientem, co umożliwia zmianę implementacji bez zmian interfejsu.



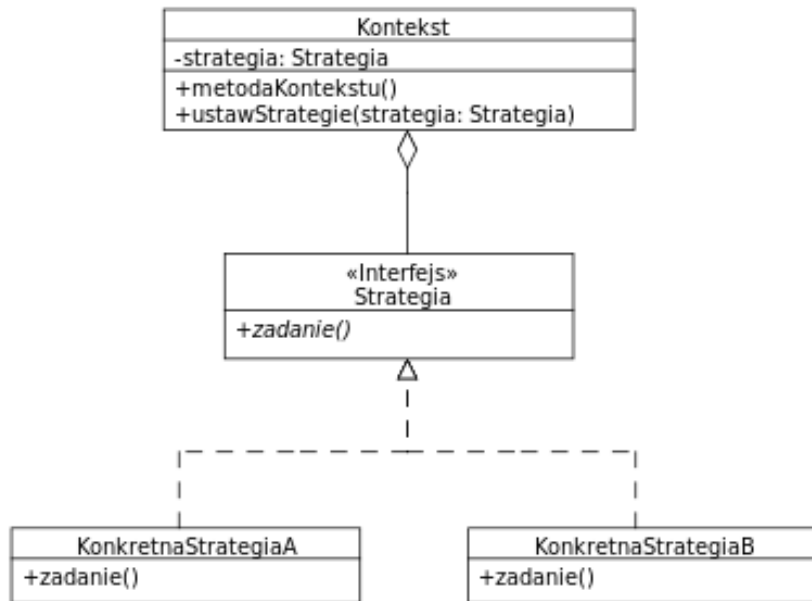
Wzorzec- Dekorator (Decorator)

Pozwala na dodanie nowej funkcji do istniejących klas dynamicznie podczas działania programu.



Wzorzec- Strategia (Strategy)

definiuje rodzinę wymiennych algorytmów i kapsułkuje je w postaci klas. Umożliwia wymienne stosowanie każdego z nich w trakcie działania aplikacji niezależnie od korzystających z nich użytkowników.

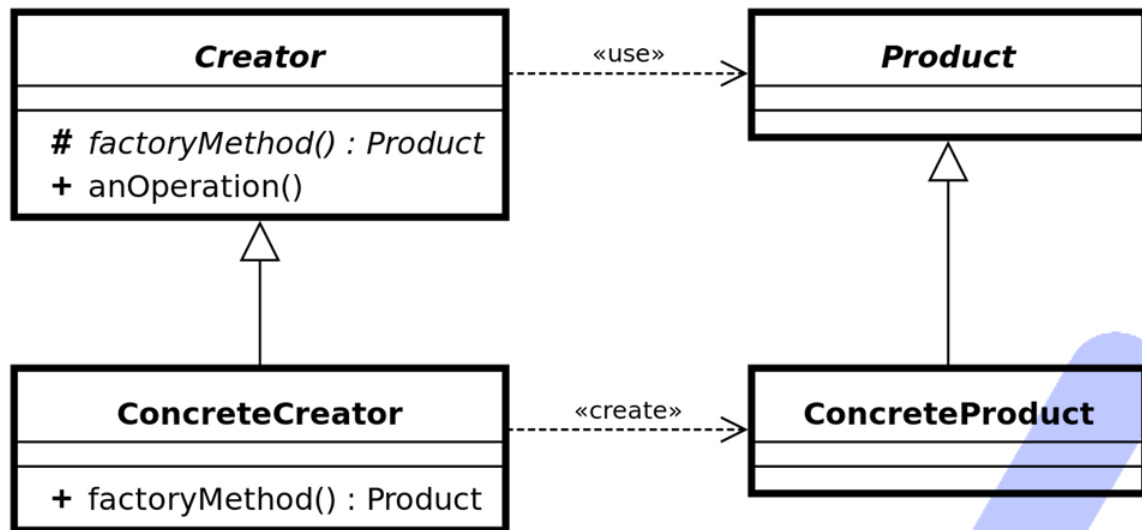


Metoda szablonowa (Template method)

Jego zadaniem jest zdefiniowanie metody, będącej szkieletem algorytmu. Algorytm ten może być następnie dokładnie definiowany w klasach pochodnych. Niezmienna część algorytmu zostaje opisana w metodzie szablonowej, której klient nie może nadpisać. W metodzie szablonowej wywoływane są inne metody, reprezentujące zmienne kroki algorytmu. Mogą one być abstrakcyjne lub definiować domyślne zachowania. Klient, który chce skorzystać z algorytmu, może wykorzystać domyślną implementację bądź może utworzyć klasę pochodną i nadpisać metody opisujące zmienne fragmenty algorytmu. Najczęściej metoda szablonowa ma widoczność publiczną, natomiast metody do przesłonięcia mają widoczność chronioną lub prywatną, tak, aby klient nie mógł ich użyć bezpośrednio.

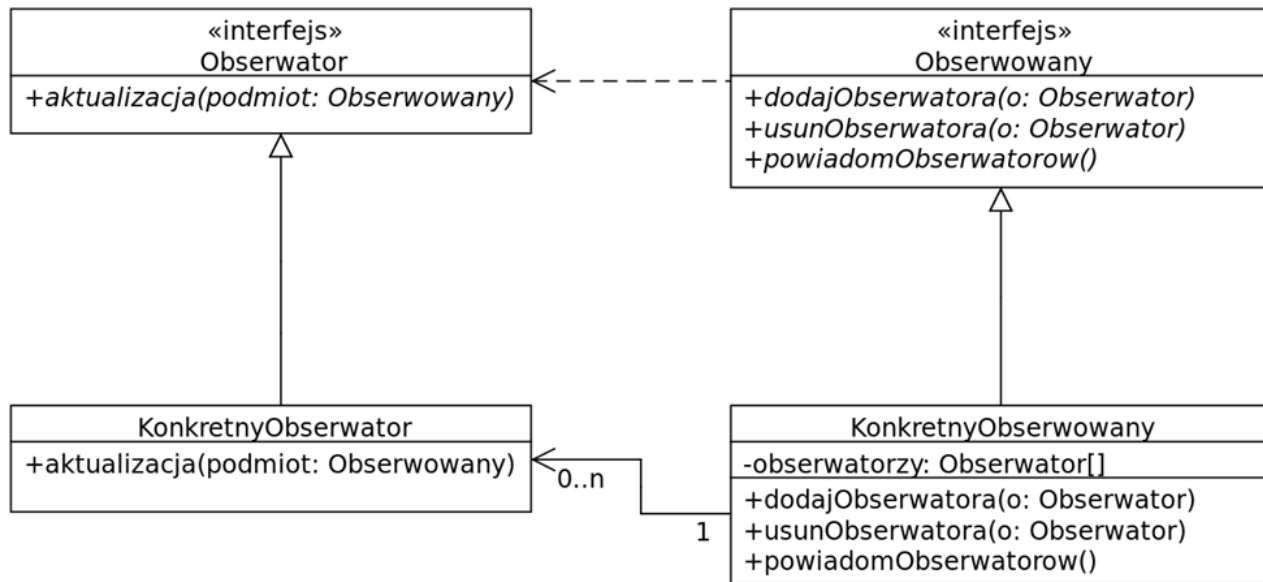
Metoda fabryczna (Factory method)

Celem tej metody jest dostarczenie interfejsu do tworzenia obiektów nieokreślonych jako powiązanych typów. Tworzeniem egzemplarzy zajmują się podklasy.



Wzorzec- Obserwator (Observer)

Używany jest do powiadamiania zainteresowane **obiekty** o zmianie stanu pewnego innego obiektu.

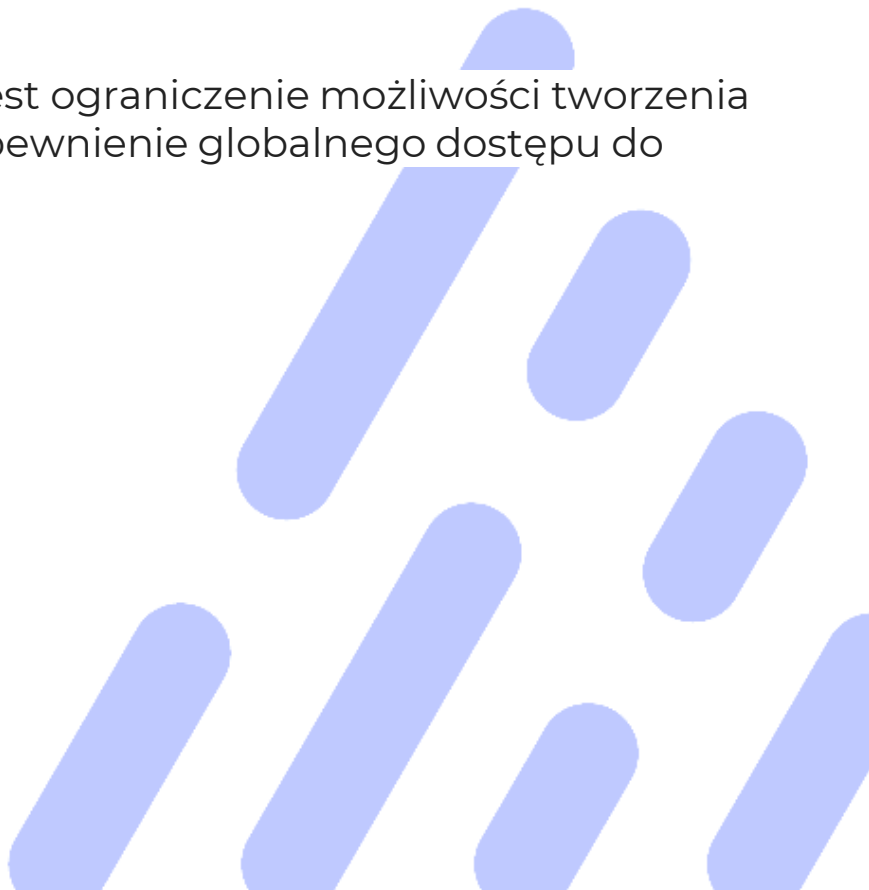


Wzorzec- Singleton

Kreacyjny wzorzec projektowy, którego celem jest ograniczenie możliwości tworzenia obiektów danej klasy do jednej instancji oraz zapewnienie globalnego dostępu do stworzonego obiektu.

Singleton

- instance : Singleton = null
- + getInstance() : Singleton
- Singleton() : void



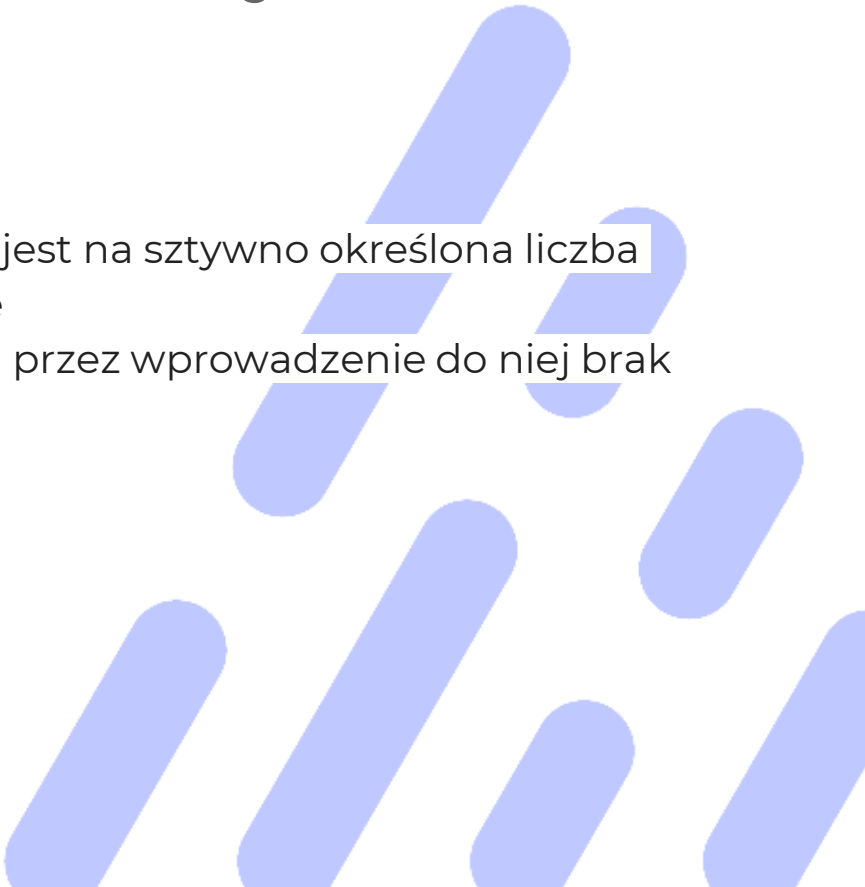
Singleton- zalety

Zalety:

- ✓ singleton nie musi ograniczać się do obsługi pojedynczej instancji klasy – przy niewielkiej zmianie podejścia można za jego pomocą zarządzać także większą liczbą obiektów,
- ✓ klasa zaimplementowana z użyciem wzorca *singleton* może samodzielnie kontrolować liczbę swoich instancji istniejących w systemie,
- ✓ proces pobierania instancji klasy jest niewidoczny dla użytkownika. Nie musi on wiedzieć, czy w chwili wywołania metody instancja istnieje czy dopiero jest tworzona,
- ✓ tworzenie nowej instancji ma charakter leniwy, tj. zachodzi dopiero przy pierwszej próbie użycia. Jeśli żaden komponent nie zdecyduje się korzystać z klasy, jej instancji nie będą niepotrzebnie przydzielone zasoby.

Singleton- wady

Wady:

- ✓ elastyczności, bo już na poziomie kodu jest na sztywno określona liczba instancji, jakie mogą istnieć w systemie
 - ✓ poważnie utrudnia testowanie aplikacji przez wprowadzenie do niej brak globalnego stanu
 - ✓ łamie zasadę jednej odpowiedzialności
 - ✓ łamie zasadę otwarte-zamknięte
 - ✓ nie można go rozszerzyć
- 

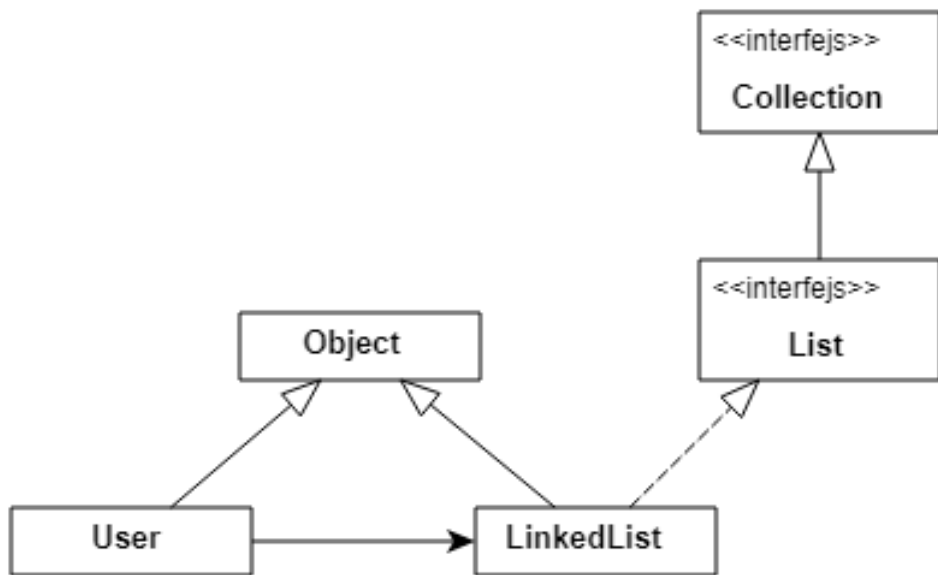
Szablon wzorca projektowego

Wzorzec projektowy jest opisany przez:

- ✓ nazwę – lakoniczny opis istoty wzorca
- ✓ klasyfikację – kategorię, do której wzorzec należy
- ✓ cel – do czego wzorzec służy
- ✓ aliasy – inne nazwy, pod którymi jest znany
- ✓ motywację – scenariusz opisujący problem i rozwiązanie
- ✓ zastosowania – sytuacje, w których wzorzec jest stosowany
- ✓ strukturę – graficzną reprezentację klas składowych wzorca
- ✓ uczestników – nazwy i odpowiedzialności klas składowych wzorca
- ✓ współdziałania – opis współpracy między uczestnikami
- ✓ konsekwencje – efekty zastosowania wzorca
- ✓ implementację – opis implementacji wzorca w danym języku
- ✓ przykład – kod stosujący wzorzec
- ✓ pokrewne wzorce – wzorce używane w podobnym kontekście

UML

UML (ang. *Unified Modeling Language*) składa się z kilkunastu rodzajów diagramów. Jest to zestaw, który pozwala na wizualną reprezentację projektu informatycznego.

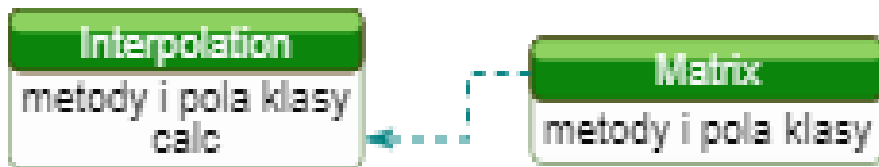


Diagramy UML

W celu zaprezentowania sposobu realizacji danego wzorca oraz zależności pomiędzy poszczególnymi obiektami w danym wzorcu stosuje się często **diagramy UML**.

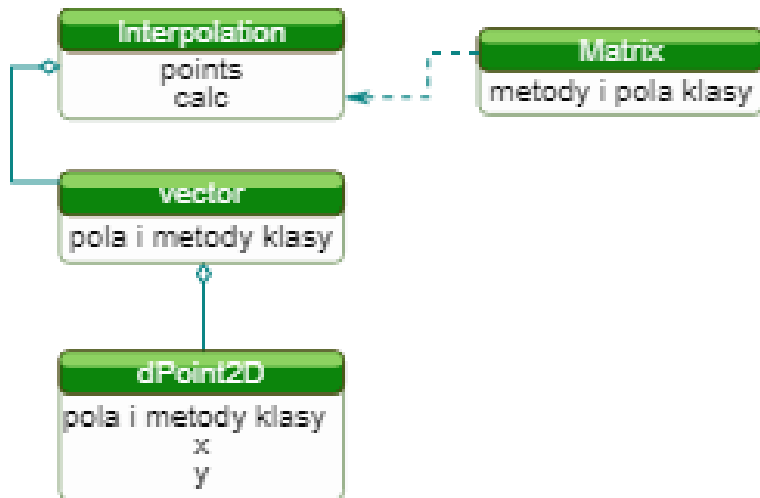
Diagramy UML opisują następujące związki (powiązania) pomiędzy obiektami klas:

- ✓ **zależność** - jest to najłagodniejszy typ zależności, w którym jedna z klas wykorzystuje drugą do realizacji części swoich zadań. Na diagramie zależność tego typu oznacza się za pomocą linii przerywanej zakończonej strzałką wskazującą kierunek zależności.



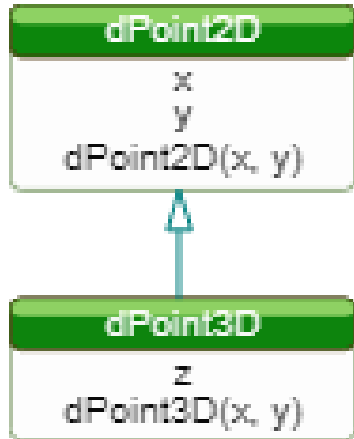
Diagramy UML c.d

- ✓ **agregacja** jest to relacja typu całość-część, gdzie elementy składowe mogą należeć do większej całości ale mogą też występować oddzielnie. Agregacja na diagramie zaznaczana jest linią zakończoną pustym rombem.



Diagramy UML c.d

- ✓ **generalizacja** jest to związek opisujący dziedziczenie przez daną klasę innych klas. Tego typu związek opisuje się na diagramie UML za pomocą niewypełnionej strzałki skierowanej w kierunku klasy dziedziczonej.



Diagramy UML c.d

- ✓ **kompozycja** jest to związek opisujący relację całość-część, gdzie element składowy stanowi integralną część całości. Każda część jest niszczone wraz z obiektem głównym. Na diagramach UML **kompozycję** oznacza się rombem zamalowanym na czarno.

Zależność	➤	Gdy jedna klasa chwilowo wykorzystuje drugą, lub wie o jej istnieniu
Asocjacja	➡	Gdy jedna klasa wykorzystuje drugą, ale nie są zależne
Agregacja częściowa	◊	Gdy klasa zawiera drugą klasę, ale współdzielili odwołanie do niej z inną
Agregacja całkowita	◆	Gdy klasa zawiera drugą klasę, i są od siebie zależne
Dziedziczenie	▷	Gdy jedna klasa jest rozszerzeniem drugiej i współdzieli swoje funkcjonalności

Czas na pytania!

Dziękuję

