WZORCE PROJEKTOWE

Dr hab. inż. Ilona Bluemke

Wzorce projektowe - architektura

C. Alexander, S. Ishikawa, M. Silverstein: A Pattern Language, Nowy York, Oxford University Press, 1977

- **„Każdy wzorzec opisuje problem, który ciągle pojawia się w naszej dziedzinie, a następnie określa zasadniczą część jego rozwiązania w taki sposób, by można było zastosować je nawet milion razy za każdym razem w nieco inny sposób”**

E.Gamma, R. Helm, R. Johnson, J.Vlissides

Design Patterns: Elements of Reusable Software, Reading, Mass., Addison-Wesley, 1995

(banda Czworga)

Wzorce w inżynierii oprogramowania

- tworząc programy spotykamy się z problemami, które rozwiązać można w podobny sposób
- w projektowaniu oprogramowania można zastosować wzorce, tak by pomagały w tworzeniu rozwiązań
- „banda czworga” - zaproponowali sposób katalogowania i opisu wzorców
- skatalogowali 23 wzorce
- postulowali wprowadzenie do projektowania obiektowego zasad i strategii opartych na wzorcach projektowych.

Powody poznawania wzorców projektowych

- wykorzystanie istniejących, wcześniej sprawdzonych wzorców przyspiesza pracę nad projektem i pozwala uniknąć błędów (nie wymyślamy rozwiązań typowych problemów)
- wzorce zapewniają wspólny punkt odniesienia, ułatwiają pracę i komunikację w zespole
- dają ogólną perspektywę widzenia, uwalniają projektanta od konieczności zbyt wczesnego zgłębiania szczegółów.

Kategorie wzorców projektowych

Strukturalne :

Do powiązania istniejących obiektów. Np.:

- *fasada, adapter* - do obsługi interfejsów,
- *most, dekorator* - do powiązania implementacji i abstrakcji.

Czynnościowe:

Do manifestacji zmiennego zachowania np.

- *Strategia* - do zawierania zmienności.

Kreacyjne:

Do utworzenia obiektów. Np.:

- *fabryka abstrakcyjna, singleton, metoda produkcyjna* – tworzenie instancji.

Metoda szablonu

Opis wzorca:

- **Intencja**: Zdefiniowanie szkieletu algorytmu i pozostawienie implementacji niektórych jego operacji klasom pochodnym. Możliwość ponownego zdefiniowania operacji bez konieczności zmieniania struktury algorytmu.
- **Problem**: Istnieje stała procedura, której poszczególne kroki mogą różnić się szczegółami.
- **Rozwiązanie**: Pozwala na zdefiniowanie zmieniających się operacji przy zachowaniu ogólnej procedury.

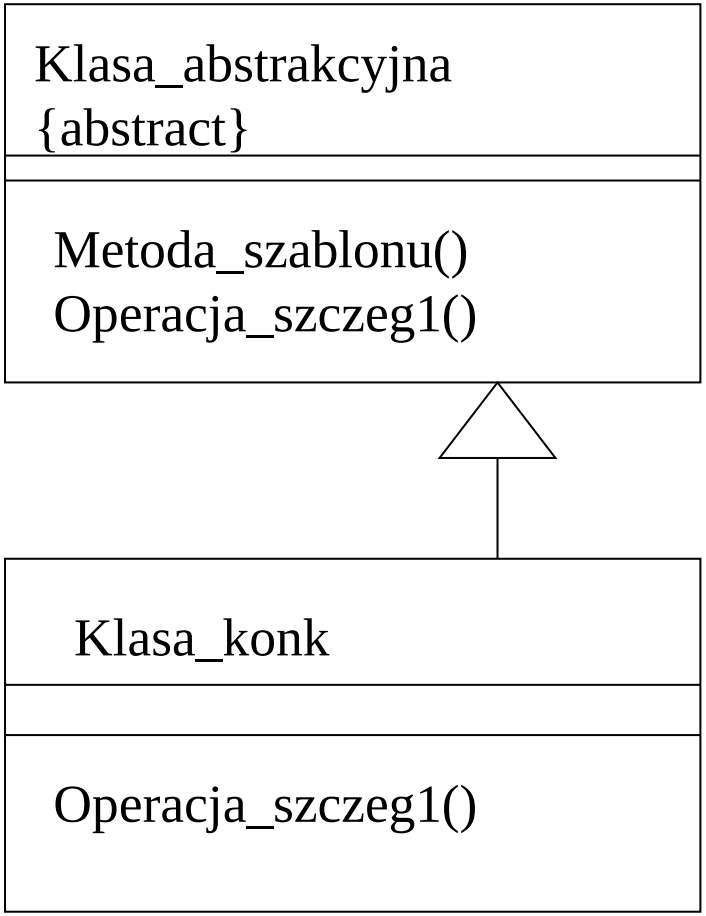
Metoda szablonu-2

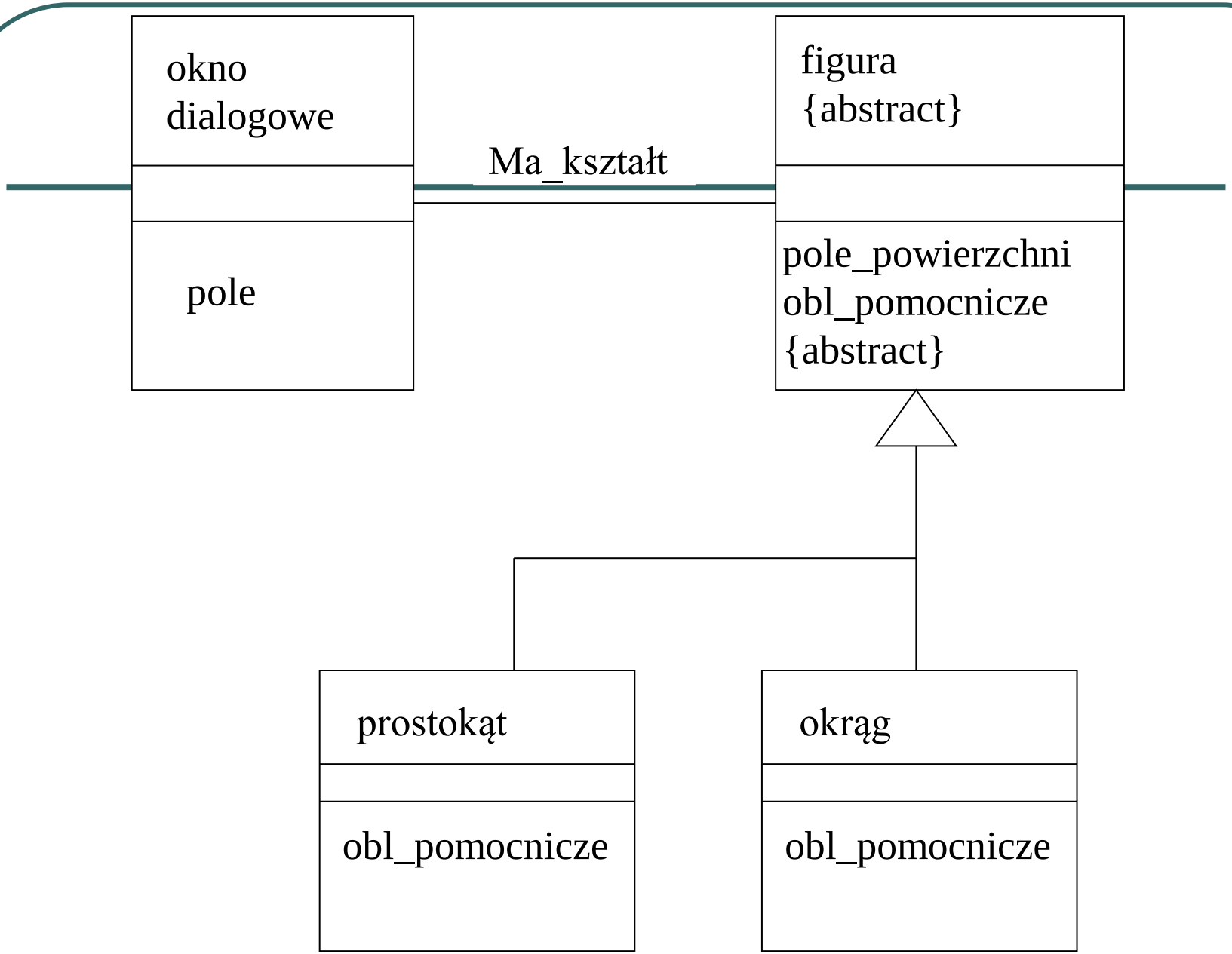
- **Uczestnicy i współpracownicy**: *Klasa_abstrakcyjna* definiuje podstawowe, wspólne zachowanie klas pochodnych w postaci metody *Metoda_szablonu* oraz szczegółowych operacji *Operacja_szczeg1*, których implementacji muszą dostarczyć klasy potomne.
- **Konsekwencje**: Szablony stanowią rozwiązanie służące powtórnemu wykorzystaniu kodu. Pomagają także zapewnić implementację określonych operacji. Wiążą one poszczególne, zmieniające się operacje wewnątrz klasy *Klasa_konk*.

Metoda szablonu-3

- **Implementacja**: Utworzenie klasy abstrakcyjnej implementującej ogólną procedurę za pomocą metod abstrakcyjnych. Implementacja tych metod musi zostać dostarczona w klasach pochodnych.
- **Referencje**: A.Shaloway, J.R. Trott „Projektowanie zorientowane obiektowo – wzorce projektowe”, Helion 2001

Metoda szablonu-4




okno
dialogowe
pole
Ma_kształt
figura
{abstract}
pole_powierzchni
obl_pomocnicze
{abstract}
prostokąt
obl_pomocnicze
okrąg
obl_pomocnicze

Metoda produkcyjna- wirtualny konstruktor

Uniezależnienie algorytmu tworzenia nowych produktów od samych produktów

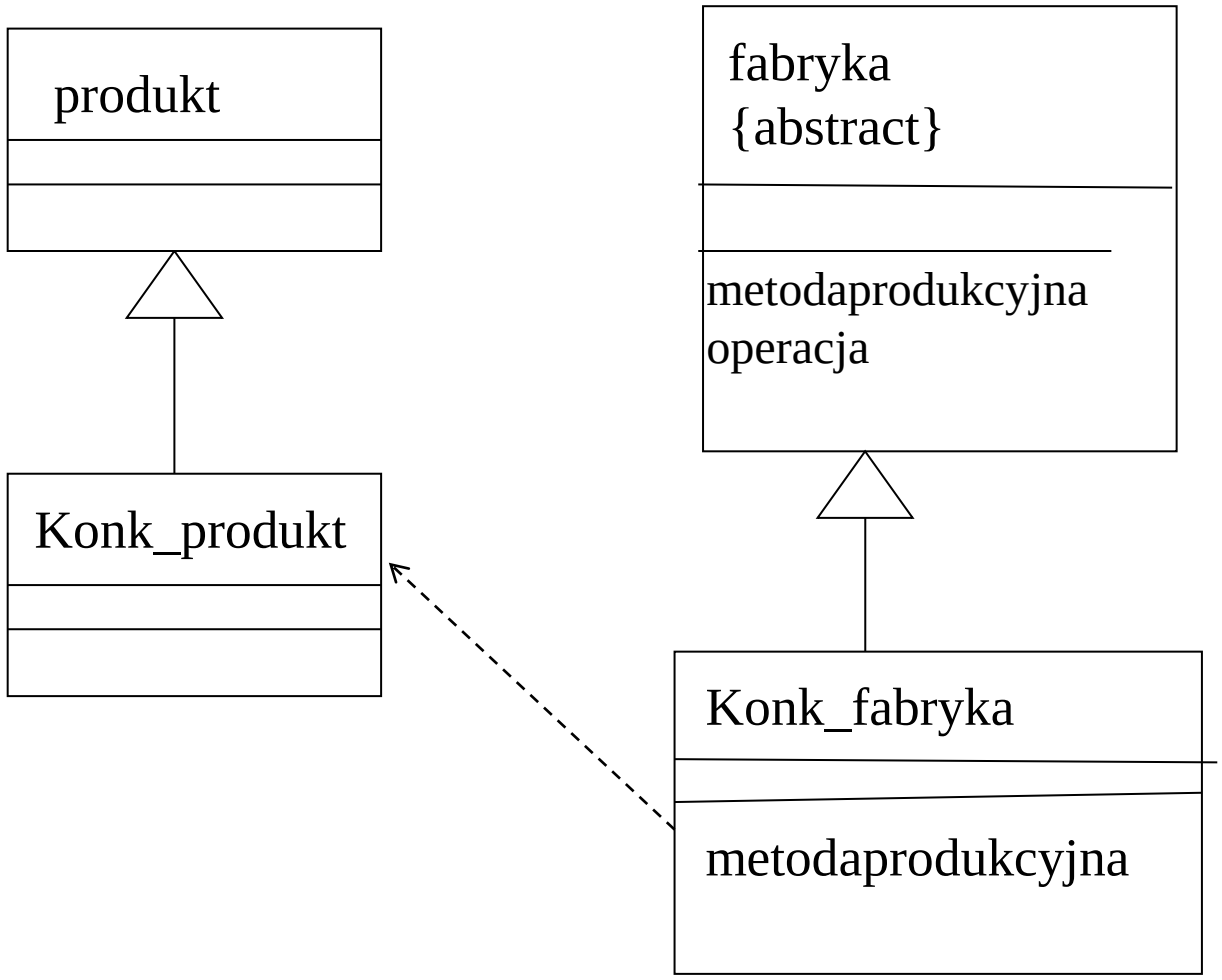
Opis wzorca:

- **Intencja**: Zdefiniowanie interfejsu służącego do stworzenia obiektu, ale pozostawienie decyzji o klasie tworzonego obiektu klasom pochodnym.
- **Problem**: Klasa musi utworzyć obiekt jednej z klas pochodnych innej klasy lecz nie wie dokładnie której. Metoda produkcyjna pozwala podjąć tę decyzję klasie pochodnej.
- **Rozwiązanie**: Klasa pochodna podejmuje decyzję o klasie i sposobie utworzenia obiektu.

Metoda produkcyjna- 2

- **Uczestnicy i współpracownicy**: *produkt* określa interfejs obiektu tworzonego przez metodę *fabryki. fabryka* definiuje interfejs zawierający metodę produkcyjną.
- **Konsekwencje**: Aby utworzyć obiekt klasy *Konk_produkt*, należy utworzyć klasę pochodną klasy *fabryka*.
- **Implementacja**: Wykorzystuje abstrakcyjną metodę klasy abstrakcyjnej (metodę wirtualną w C++). Klasa abstrakcyjna używa tej metody, kiedy musi utworzyć odpowiedni obiekt, ale nie wie jaki obiekt będzie w rzeczywistości utworzony.

Metoda produkcyjna-3



Metoda produkcyjna-4

- Nowy produkt definiowany jako podklasa klasy *produkt*, tworzona jest konkretna fabryka odpowiedzialna za jego tworzenie.
- Algorytm produkcji w metodzie *fabryka.operacja* (produkt:=metodaprodukcyjna).
- *Konk_fabryka.metodaprodukcyjna* zwraca nowy *Konk_produkt*. Wadą jest tworzenie nowej podklasy dla każdego rodzaju produktu

Fabryka abstrakcyjna

Wzorzec fabryka abstrakcyjna jest wykorzystywany w sytuacjach, gdy wybór określonej konfiguracji powinien powodować tworzenie grupy określonych produktów.

Dodanie nowej konfiguracji to zdefiniowanie nowej konkretnej fabryki oraz grupy nowych produktów. W trakcie działania systemu wykorzystywana jest tylko jedna konkretna fabryka. Wybór tej klasy nie ma wpływu na żądania kierowane przez klienta, poprzez dziedziczenie interfejsu.

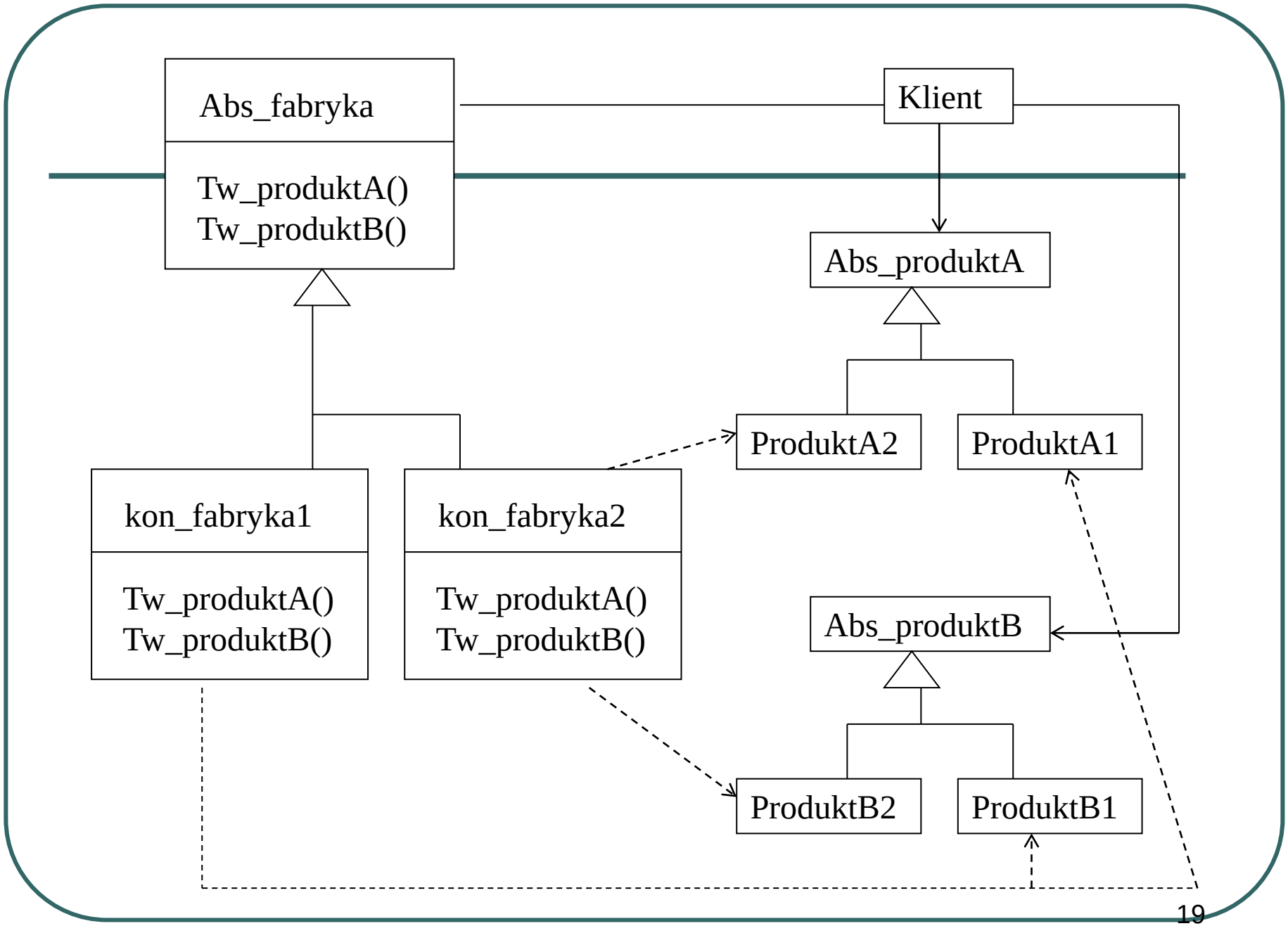
Fabryka abstrakcyjna - 2

Opis wzorca:

- **Intencja**: Uzyskanie rodzin obiektów właściwych w określonym przypadku
- **Problem**: Utworzenie odpowiednich rodzin obiektów
- **Rozwiązanie**: Koordynuje utworzenie rodzin obiektów. Podsuwa sposób pozwalający wydzielić z obiektów użytkownika reguły tworzenia obiektów, które są przez nie używane.
- **Uczestnicy i współpracownicy**: *Abs_ Fabryka* definiuje interfejs określający sposób utworzenia każdgo z obiektów danej rodziny. Typowo każda z rodzin obiektów posiada własną klasę *kon_fabryka*.

Fabryka abstrakcyjna - 3

- **Konsekwencje**: Wzorzec izoluje reguły opisujące sposób wykorzystania obiektów od reguł decydujących o utworzeniu tych obiektów.
- **Implementacja**: Definiuje klasę abstrakcyjną specyfikującą tworzone obiekty. Dla każdej z rodzin obiektów implementuje się klasę konkretną. W celu dokonania wyboru tworzonych obiektów mogą być zastosowane pliki konfiguracyjne lub tabela bazy danych.
- **Referencje**: A.Shaloway, J.R. Trott „Projektowanie zorientowane obiektowo – wzorce projektowe”, Helion 2001


Abs_fabryka
Tw_produktA()
Tw_produktB()
Klient
Abs_produktA
ProduktA2
ProduktA1
kon_fabryka1
Tw_produktA()
Tw_produktB()
kon_fabryka2
Tw_produktA()
Tw_produktB()
Abs_produktB
ProduktB2
ProduktB1

Adapter

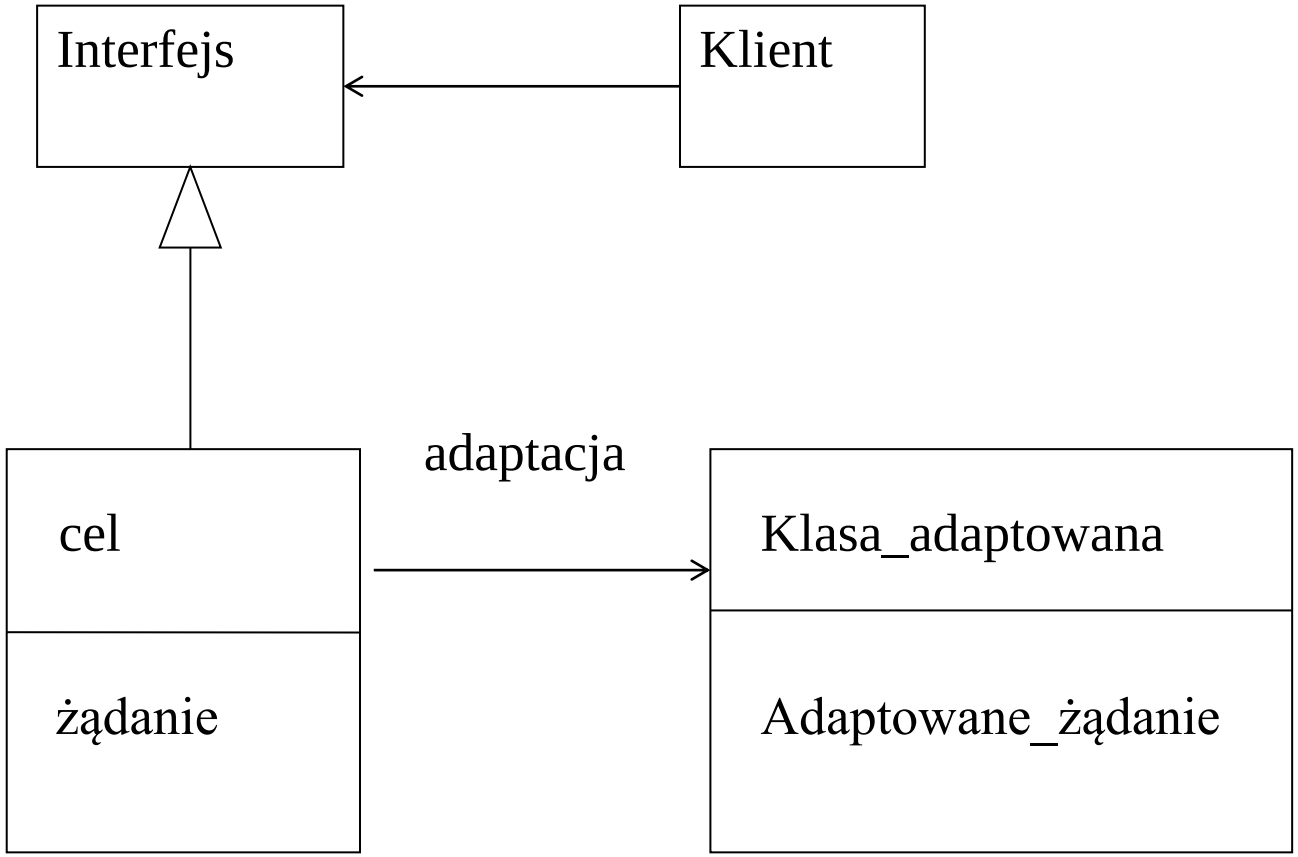
Opis wzorca:

- **Intencja**: dopasowanie istniejącego obiektu (nad którym programista nie ma kontroli) do określonego sposobu wywołania.
- **Problem**: Obiekt przechowuje potrzebne dane oraz zachowuje się w pożądany sposób, jednak posiada nieodpowiedni interfejs. Często włączamy go do hierarchii dziedziczenia pewnej klasy abstrakcyjnej posiadanej lub definiowanej.
- **Rozwiązanie**: Adapter obudowuje obiekt pożądanym interfejsem

Adapter - 2

- **Uczestnicy i współpracownicy**: Adapter dostosowuje interfejs klasy *klasa-adaptowana* tak, by był zgodny z interfejsem klasy bazowej *cel* Umożliwia to użytkownikowi wykorzystanie obiektu klasy *klasa-adaptowana* oraz instancji klasy *cel*.
- **Konsekwencje**: Zastosowanie wzorca adapter pozwala dopasować istniejące obiekty do tworzonych struktur klas i uniknąć ograniczeń związanych z ich interfejsem.
- **Implementacja**: Zawiera istniejąca klasę w nowej klasie, która posiada wymagany interfejs i wywołuje odpowiednie metody zawieranej klasy

Adapter - 3


Interfejs
Klient
adaptacja
cel
żądanie
Klasa_adaptowana
Adaptowane_żądanie

Adapter - 4

- Interfejs jest definiowany w abstrakcyjnej klasie ***Interfejs*** a jego funkcjonalność w podklasach. Metody klasy ***cel*** mogą dokonywać konwersji typów lub tworzyć nowe algorytmy określające funkcjonalność nie przewidzianą w adaptowanych klasach.
- Wywołanie metody ***klasy-adaptowanej*** jest dokonywane przez trawersowanie związku.

Strategia

Wzorzec jest stosowany w celu ukrycia szczegółów algorytmu przed klientem, konfigurowania różnych zachowań klas.

W metodzie klasy abstrakcyjnej można umieścić wspólne części algorytmu, w podklasach ich specjalizację.

Opis wzorca:

- **Intencja**: Umożliwia użycie różnych wersji algorytmu czy reguł biznesowych w zależności od kontekstu

Strategia -2

- **Problem**: Wybór algorytmu zależy od używającego go obiektu lub danych na których operuje. Jeśli algorytm nie zmienia się wzorzec nie jest potrzebny.
- **Rozwiązanie**: Separuje wybór wersji algorytmu od jego implementacji. Umożliwia wybór algorytmu na podstawie kontekstu.

Strategia -3

- **Uczestnicy i współpracownicy**:

strategia specyfikuje sposób użycia różnych algorytmów.

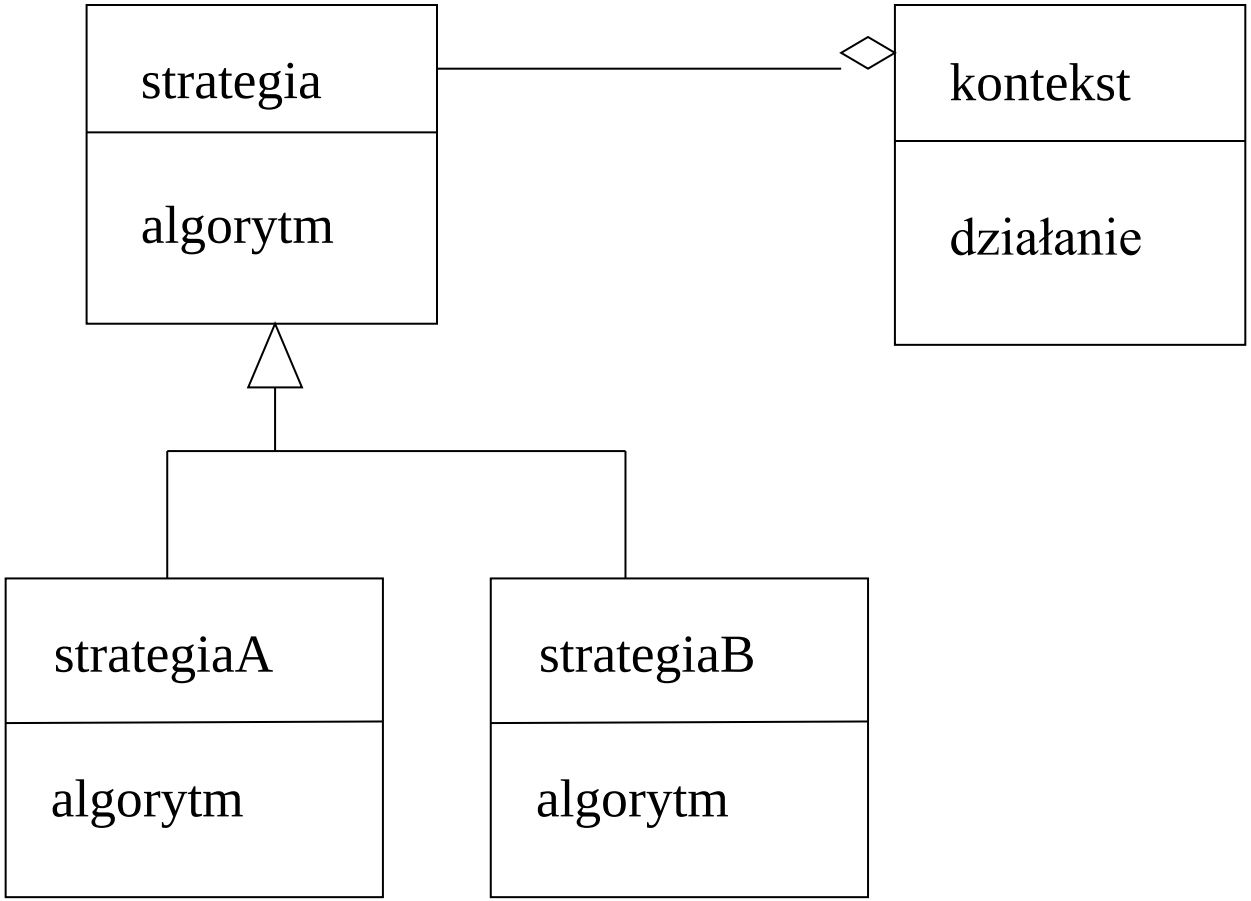
strategiaA, *strategia*B implementują konkretny algorytm.

kontekst korzysta z obiektu klasy *strategiaA* czy *strategia*B odwołując się do niego jak do obiektu klasy *strategia. strategia i kontekst* współdziałają w wyborze algorytmu, czasami *strategia* kieruje w tym celu żądanie do klasy *kontekst*. *kontekst* przekazuje żądania korzystających z niego obiektów klasie *strategia*.

Strategia - 4

- **Konsekwencje**: Strategia definiuje rodzinę algorytmów. Instrukcje wyboru mogą zostać wyeliminowane. Wszystkie algorytmy muszą być wywoływane w ten sam sposób.
- **Implementacja**: Klasa, która używa algorytmu *kontekst* zawiera klasę abstrakcyjną s*trategia,* która posiada metodę abstrakcyjną określającą sposób wywołania algorytmu. Każda z jej klas pochodnych implementuje algorytm we własnym zakresie. W przypadku, gdy metody mają wspólną część, metoda ta nie musie być abstrakcyjna.
- **Referencje**: A.Shaloway, J.R. Trott „Projektowanie zorientowane obiektowo – wzorce projektowe”, Helion 2001

Strategia - 5


strategia
algorytm
kontekst
działanie
strategiaA
algorytm
strategiaB
algorytm

Obserwator

Wzorzec stosowany przy projektowaniu środowisk prezentacji danych.

- **Opis wzorca**:
- **Intencja**: Definiuje „jeden-do-wielu” zależności między obiektami. Zmiana stanu jednego obiektu automatycznie propaguje na związane z nim obiekty.
- **Problem**: Należy powiadomić zmieniającą się listę obiektów o pewnym zdarzeniu.
- **Rozwiązanie**: Obiekty klasy *obserwator* przekazują odpowiedzialność za monitorowane zdarzenia centralnemu obiektowi *dana*.

Obserwator -2

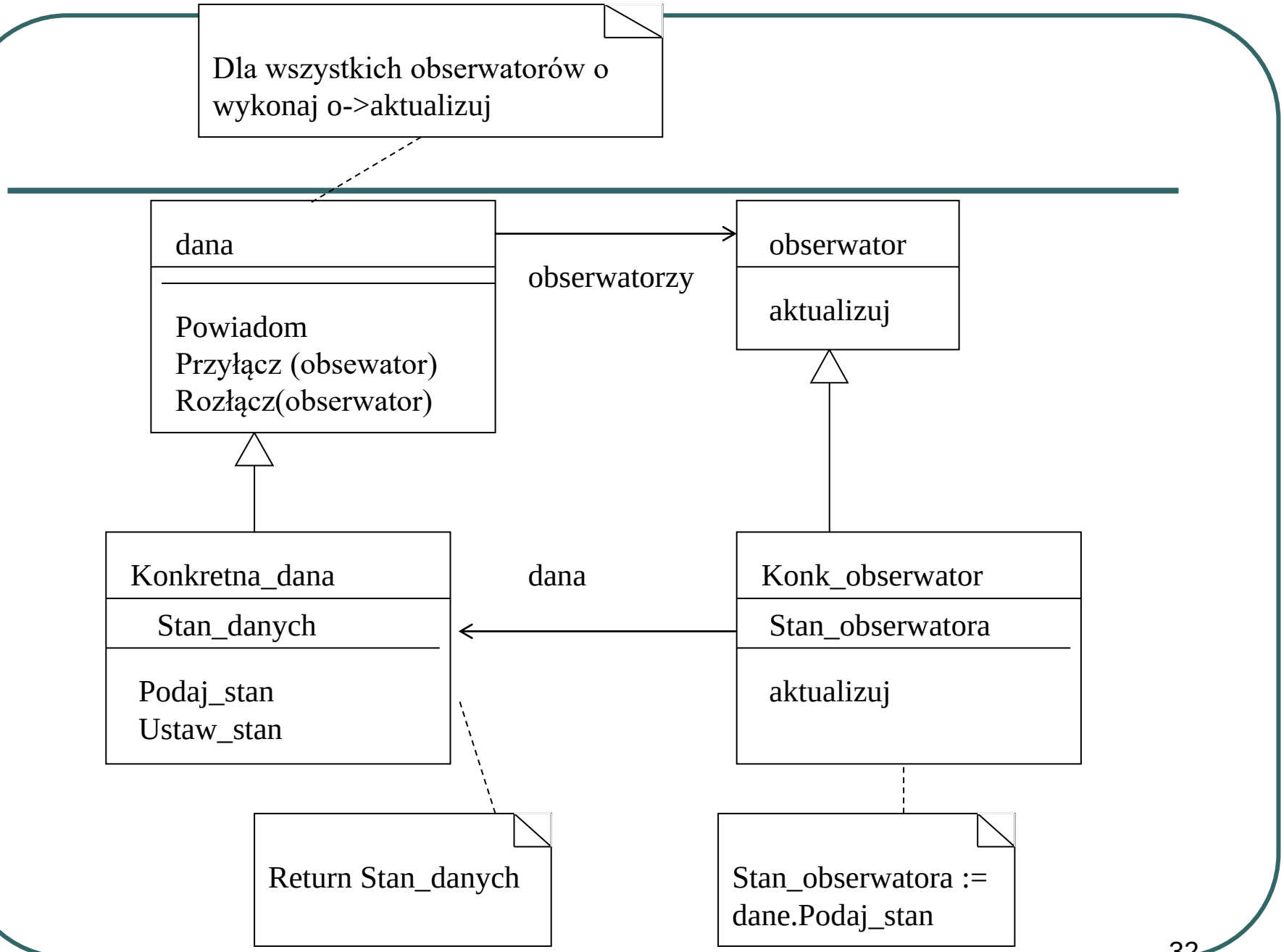
- **Uczestnicy i współpracownicy**: *dana* wie, które obiekty klasy *obserwator* należy zawiadomić, ponieważ rejestrują się u niego. *dana* zawiadamia obiekty klasy *obserwator* o wystąpieniu zdarzenia. Obiekty klasy *obserwator* są odpowiedzialne za zarejestrowanie się u obiektu *dana* i uzyskanie od niego potrzebnej informacji, gdy zostaną powiadomione o zdarzeniu.
- **Konsekwencje**: *dana* może niepotrzebnie powiadamiać o zdarzeniu różne rodzaje obserwatorów nawet, wtedy gdy są zainteresowane jego wystąpieniem tylko w pewnych przypadkach.

Obserwator -3

- **Implementacja**: Tworzy obiekty klasy ***obserwator***, które mają być informowane o wystąpieniu pewnego zdarzenia i w tym celu rejestrują się u obiektu klasy *dana*.

Kiedy zachodzi zdarzenie, ***dana*** powiadamia zarejestrowane obiekty klasy ***obserwator***.

- **Referencje**: A.Shaloway, J.R. Trott „Projektowanie zorientowane obiektowo – wzorce projektowe”, Helion 2001


Dla wszystkich obserwatorów o
wykonaj o->aktualizuj
dana
Powiadom
Przyłącz (obsewator)
Rozłącz(obserwator)
obserwatorzy
obserwator
aktualizuj
Konkretna_dana
Stan_danych
Podaj_stan
Ustaw_stan
dana
Konk_obserwator
Stan_obserwatora
aktualizuj
Return Stan_danych
Stan_obserwatora :=
dane.Podaj_stan

Obserwator -5

- Klasa ***dane*** – zna swoich obserwatorów, dowolna liczba obserwatorów może obserwować dane. Obserwatorzy są rejestrowani – ***przyłącz, rozłącz*** (wskazanie na klasę). Metoda *powiadom* informuje wszystkie zarejestrowane obiekty o zmianach wewnątrz niej.
- Klasa ***obserwator*** – definiuje interfejs dla obiektów, które mają być powiadamiane o zmianach w danych. Obserwator po otrzymaniu informacji o konieczności aktualizacji wysyła żądanie do klasy *dane* z prośbą o podania aktualnego stanu. Powiązanie na poziomie klas konkretnych daje możliwość komunikacji od klas obserwujących do danych.
- Klasa ***Konk_obserwator*** – zawiera wskazanie na konkretne dane oraz stan, który powinien być spójny z danymi.
- Klasa ***Konkretne_dane*** – powiadamia obserwatorów o zmianie stanu, przechowuje stan

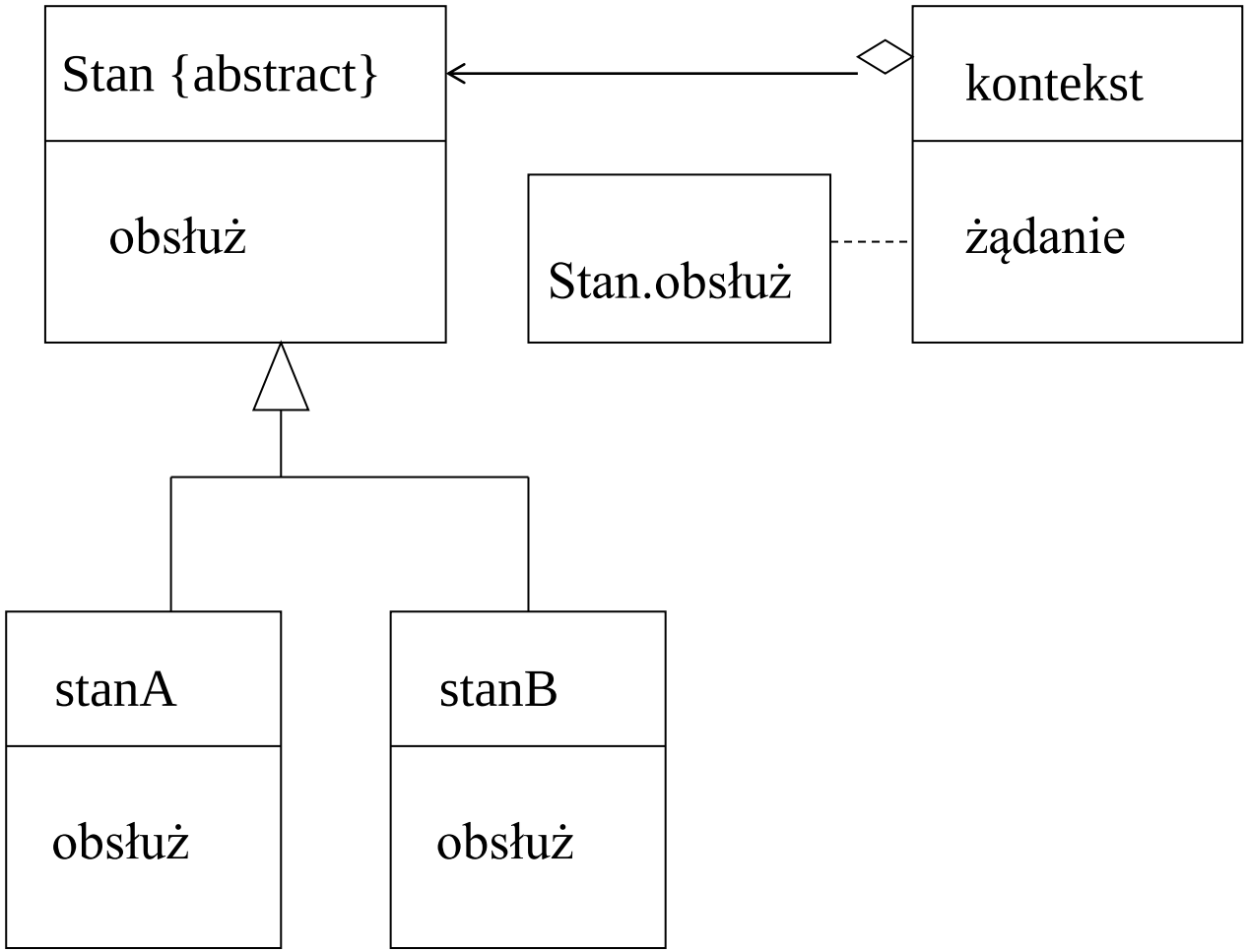
Stan

Wzorzec stan umożliwia wygodną implementację informacji zawartych w diagramie stanów klasy.

Ułatwia wprowadzanie zmian zachowania, są to zmiany prostych metod klas implementujących stany.

Wadą jest tworzenie rozbudowanej hierarchii klas.

Stan - 2


Stan {abstract}
obsłuż
kontekst
żądanie
Stan.obsłuż
stanA
obsłuż
stanB
obsłuż

Stan - 3

- Abstrakcyjna klasa ***Stan*** definiuje interfejs dla wszystkich klas pochodnych definiujących zachowanie klasy w określonym stanie.
- Klasa ***kontekst*** udostępnia użytkownikowi usługi. Wykonanie usługi uzależnionej od stanu jest w odpowiedniej klasie.
- Np. TCPConnection, TCPState, TCPestablished, TCPlisten, TCPclosed

Ambasador

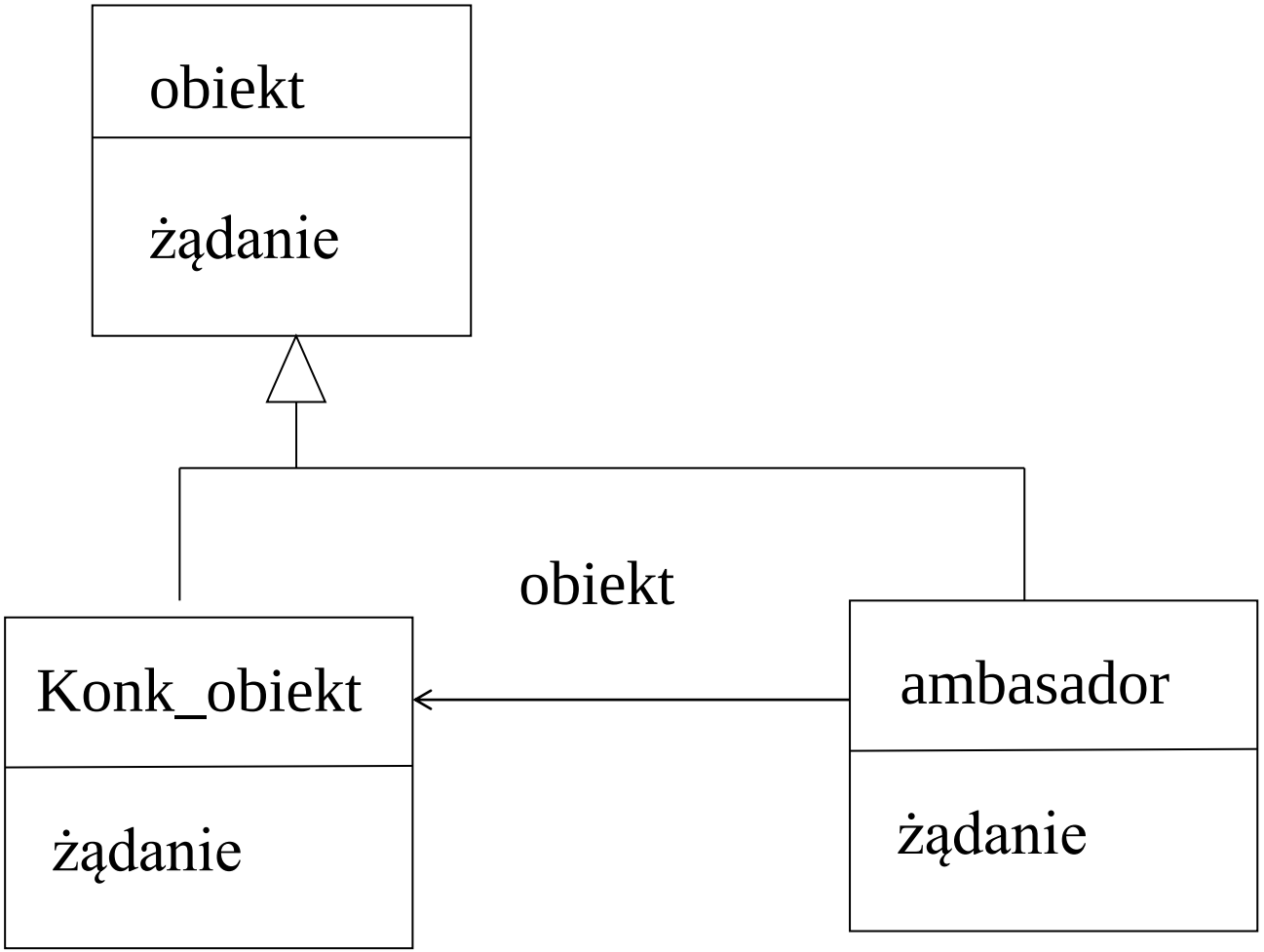
Wykorzystywany w sytuacjach, gdy nie jest konieczne stałe utrzymywanie zainicjowanego konkretnego obiektu w systemie. Rzeczywiste tworzenie obiektu jest opóźnione.

Ambasador ma identyczny interfejs jak reprezentowany przez niego obiekt.

Ambasador powołuje do życia konkretny obiekt.

Wzorzec ambasador stosowany w sytuacjach, gdy utrzymywanie zainicjowanego obiektu powoduje duże zużycie zasobów wpływające na efektywność pracy aplikacji..

Ambasador -2


obiekt
żądanie
obiekt
Konk_obiekt
żądanie
ambasador
żądanie