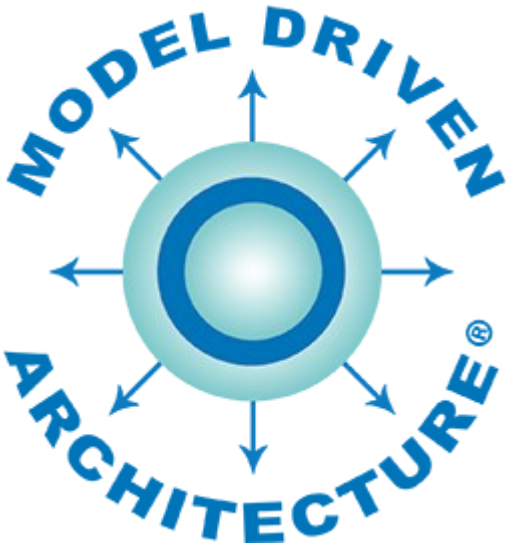


Architektura Sterowana Modelem

Model Driven Architecture

prezentacja 6

Wzorce projektowe w budowie modelu oprogramowania

wersja 1.0

dr inż. Paweł Głuchowski

Wydział Informatyki i Telekomunikacji, Politechnika Wrocławska

Treść prezentacji

1. Wzorzec projektowy
2. Prosty model analizy
3. 5-warstwowy wzorzec architektury
4. Wzorzec architektury MVC
5. Wzorzec architektury MVP
6. Inne warstwowe wzorce architektury
7. Wzorce projektowe kreacyjne
8. Wzorce projektowe strukturalne
9. Wzorce projektowe czynnościowe
10. Relacje między wzorcami projektowymi
11. Złożone wzorce projektowe

1

Wzorzec projektowy

Wzorzec projektowy

- **Koncepcyjny model rozwiązania powtarzającego się problemu projektowego.**
  - Koncepcja, NIE algorytm, NIE implementacja.
  - Wymaga dostosowania do konkretnego problemu.
  - Opisuje zależności między warstwami oprogramowania lub jego klasami.
  - Standaryzuje strukturę oprogramowania – poprawia jakość kodu.
  - Ułatwia późniejsze zmiany.
- **Dobra praktyka stosowania wzorców projektowych:**
  - Stosuje się je na początku modelowania struktury systemu, aby ułatwić późniejsze zmiany.
  - Im łatwiej rozpoznać zastosowany wzorzec, tym łatwiej zrozumieć strukturę oprogramowania.

Opis wzorca

- **Nazwa** wzorca.
- **Przeznaczenie** – krótki opis problemu i jego rozwiązania.
- **Motywacja** – dokładny opis rozwiązania problemu (scenariusz) i kontekst jego zastosowania.
- **Stosowalność** – kiedy i gdzie można zastosować wzorzec.
- **Struktura** – uczestnicy wzorca i relacje między nimi (np. diagram klas).
- **Konsekwencje** – główne i uboczne skutki zastosowania wzorca.
- **Implementacja** – wskazówki do implementacji wzorca.
- **Przykład kodu** w wybranym języku programowania.
- **Pokrewne wzorce**.

Wzorce architektury

- **Architektura wielowarstwowa** /multi-tier architecture/
  - SI składa się z połączonych warstw (np. system *klient-serwer*).
  - Podział funkcjonalny – każda warstwa pełni inne zadania,
  - Podział fizyczny – każda warstwa jest osobnym komponentem systemu.
  - Ułatwia stosowanie wzorców prostych w ramach danej warstwy.
- **Architektura 3-warstwowa** /3-tier architecture/
  - Podział na 3 warstwy: prezentacji, biznesową i zasobów zwany **BCE** (Boundary – Control – Entity):
  - **MVC** (Model – Widok – Kontroler),
  - **MVP** (Model – Widok – Prezenter),
  - **MVP** (Model – View – Presenter),
  - **PAC** (Presentation – Abstraction – Control),
  - **HMVC** (Hierarchical Model – View – Controller),
  - **MVVM** (Model – View – Viewmodel),
  - **MVA** (Model – View – Adapter).

Wzorce projektowe

- Wiążą ze sobą klasy (zwykle w ramach jednej warstwy oprogramowania):
  - **Kreacyjny** – opisuje sposób tworzenia, inicjacji i konfiguracji klas i obiektów (ułatwia ponowne użycie kodu).
  - **Strukturalny** – opisuje struktury złożone z powiązanych ze sobą klas i obiektów.
  - **Czynnościowy** – opisuje zachowanie, interakcję i odpowiedzialność współpracujących ze sobą klas i obiektów (podział zadań).
  - **Antywzorzec** – opisuje powtarzające się ZŁE rozwiązanie problemu.

Inne wzorce

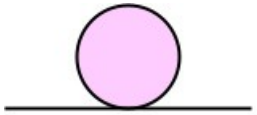
- **Klasowy** – opisuje statyczne relacje między klasami.
- **Obiektowy** – opisuje dynamiczne relacje między obiektami.

2

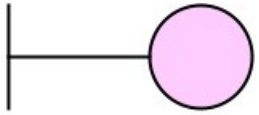
Prosty model analizy

Prosty model analizy

- Pokazuje zastosowanie klasy w fazie koncepcji
  - abstrakcja niezależna od implementacji.
- Stosowany do wzorców projektowych BCE, m.in.:
  - MVC (Model – Widok – Kontroler),
  - MVP (Model – Widok – Prezenter).
- Stosowana na diagramach klas, komunikacji i sekwencji.



Klasa Encji

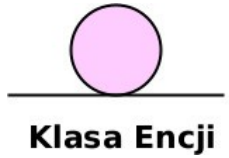


Klasa graniczna

Klasa sterująca

Prosty model analizy

Klasa encji /entity/



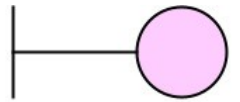
- **Modeluje dane** (informacje) lub struktury danych:
  - używane i przetwarzane przez SI,
  - trwale lub długo istniejące w modelowanym świecie.
- Może posiadać operacje zmieniające swój stan.

- Wzorce projektowe MVC i MVP: warstwa modelu.

- Stereotyp klasy: **«*entity*»**.

Prosty model analizy

Klasa graniczna /boundary/



- **Modeluje interfejs użytkownika** i inne interfejsy:
  - pośredniczy między systemem a (zwykle 1) aktorem,
  - oddziela system od aktora.
- Może być częścią GUI – abstrakcją okna lub jego elementu.
- Może być interfejsem API modelowanego lub zewnętrznego systemu.

- Wzorce projektowe MVC i MVP: warstwa widoku.
  - Niektóre wzorce: warstwa modelu – dedykowany interfejs do modelu.

- Stereotyp klasy: «***boundary***».

Klasa sterująca /control/

- **Steruje działaniem**:
  - logiką biznesową, dynamiką systemu, realizacją przypadku użycia;
  - przepływem sterowania i danych między klasami tej samej i różnych warstw.
- **Rozdziela warstwy oprogramowania:**
  - warstwę integracji od warstwy biznesowej,
  - warstwę biznesową od warstwy prezentacji,
  - warstwę modelu od warstwy widoku.

- Wzorce projektowe MVC i MVP: warstwy kontrolera / prezentera.

- Stereotyp klasy: «***control***».

3

5-warstwowy wzorzec architektury

5-warstwowy model logicznego rozdzielenia zadań

na podst. „J2EE. Wzorce projektowe”, D. Alur, J. Crupi, D. Malks, wyd. Helion 2004

- **Warstwa klienta** – interakcja z użytkownikiem, prezentacja GUI.
- **Warstwa prezentacji** – zarządzanie sesją, tworzenie i prezentacja danych.
- **Warstwa biznesowa** – obiektowy model danych i logika biznesowa.
- **Warstwa integracji** – dostęp do danych i zewnętrznych systemów.
- **Warstwa zasobów** – zasoby, dane i zewnętrzne usługi.

5-warstwowy model logicznego rozdzielenia zadań

- **Warstwa klienta**
  - Klientem SI jest jego użytkownik: człowiek, urządzenie, usługa internetowa, rozbudowana aplikacja kliencka itp.
- **Warstwa prezentacji**
  - Udostępnia użytkownikowi graficzny interfejs SI (GUI).
  - Udostępnia użytkownikowi usługi biznesowe komponentów warstwy biznesowej w sposób dla niego zrozumiały.
- **Warstwa biznesowa**
  - Udostępnia komponentom warstwy prezentacji usługi biznesowe przetwarzane przez nie dane.
  - Ogranicza relacje między klientem a usługami biznesowymi.
  - Ukrywa szczegóły implementacji operacji na danych (usługi biznesowe) i strukturę danych.

5-warstwowy model logicznego rozdzielenia zadań

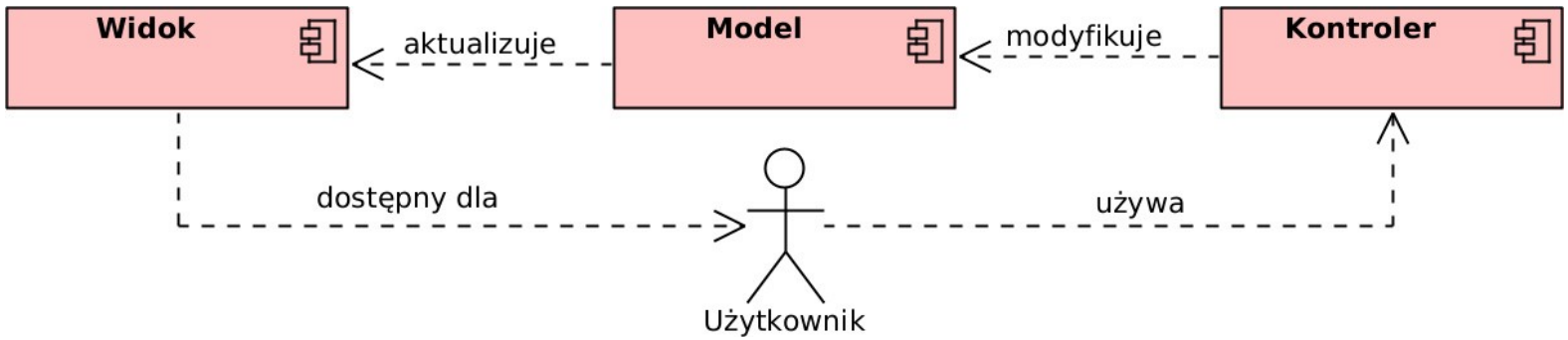
- **Warstwa integracji**
  - Udostępnia komponentom warstwy biznesowej dostęp do trwałych danych z warstwy zasobów.
  - Korzysta z **DataAccessObject** (obiekt dostępu do danych):
    - zapewnia jednakowy dostęp do danych z różnych źródeł,
    - ukrywa implementację dostępu do danych,
    - implementuje operacje CRUD (Create, Read, Update, Delete).
- **Warstwa zasobów**
  - Źródło trwałych danych, np. baza danych, zewnętrzna usługa.

4

Wzorzec architektury MVC

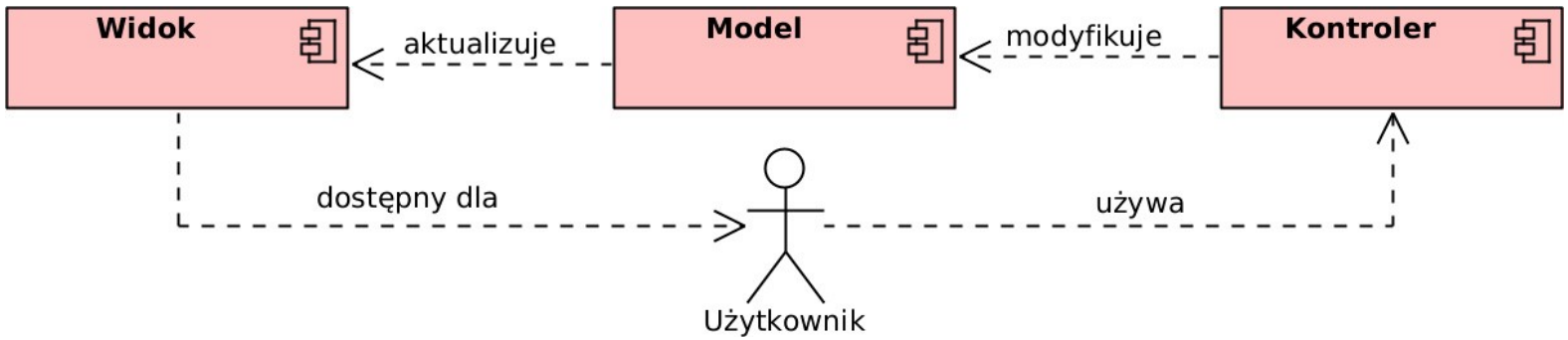
MVC (Model – View – Controller)

- 3 warstwy oprogramowania:
  - **model** /model/ – warstwa modelu,
  - **widok** /view/ – warstwa prezentacji,
  - **kontroler** /controller/ – warstwa biznesowa.
- Idea działania:
  - Użytkownik używa Kontrolera,
  - Kontroler modyfikuje Model,
  - Model aktualizuje Widok,
  - Widok jest dostępny dla Użytkownika.



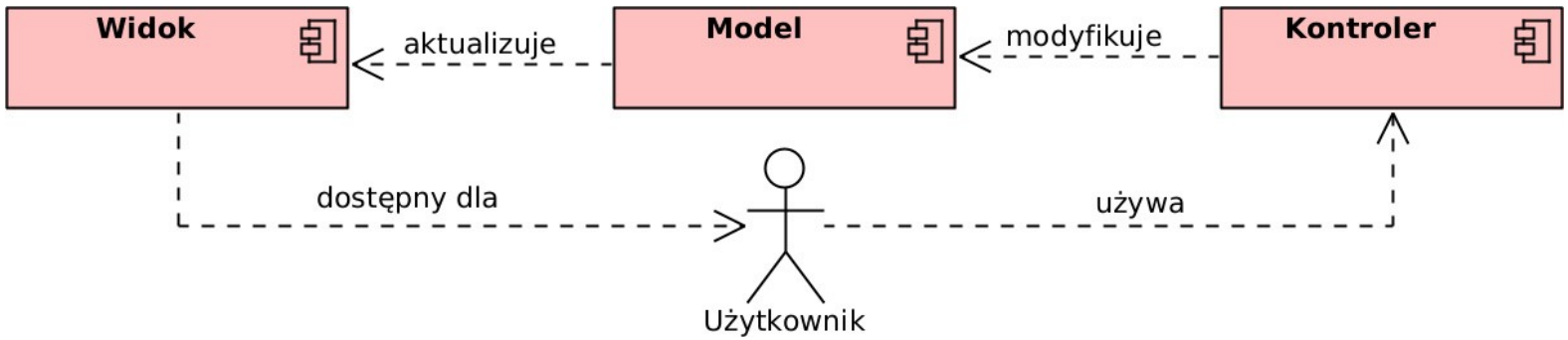
Kontroler

- Steruje przepływem danych z i do modelu.
- Steruje (pośrednio przez model) aktualizacją widoku.
- Reaguje na działania użytkownika
  (np. wywołuje operację obiektu z modelu).
- Pobiera i przetwarza dane od użytkownika.



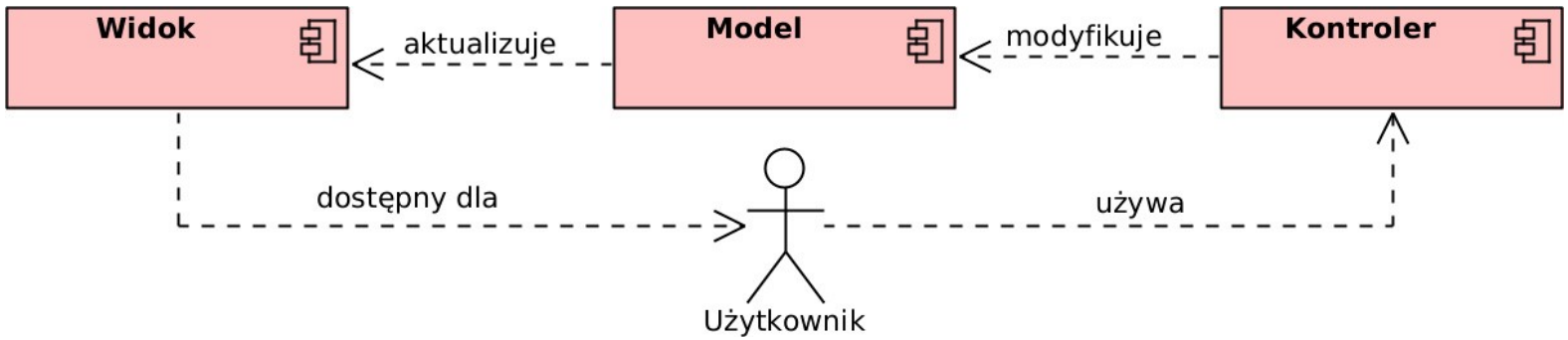
Model

- Definiuje obiekty przechowujące dane i operacje ich przetwarzania.
- Jest samodzielny i niezależny od widoku.
- Może być pasywny (obiekt sam nie zmienia swego stanu).
- Może być aktywny (obiekt sam może zmienić swój stan, niezależnie od kontrolera).
- NIE definiuje sposobu prezentacji danych użytkownikowi.



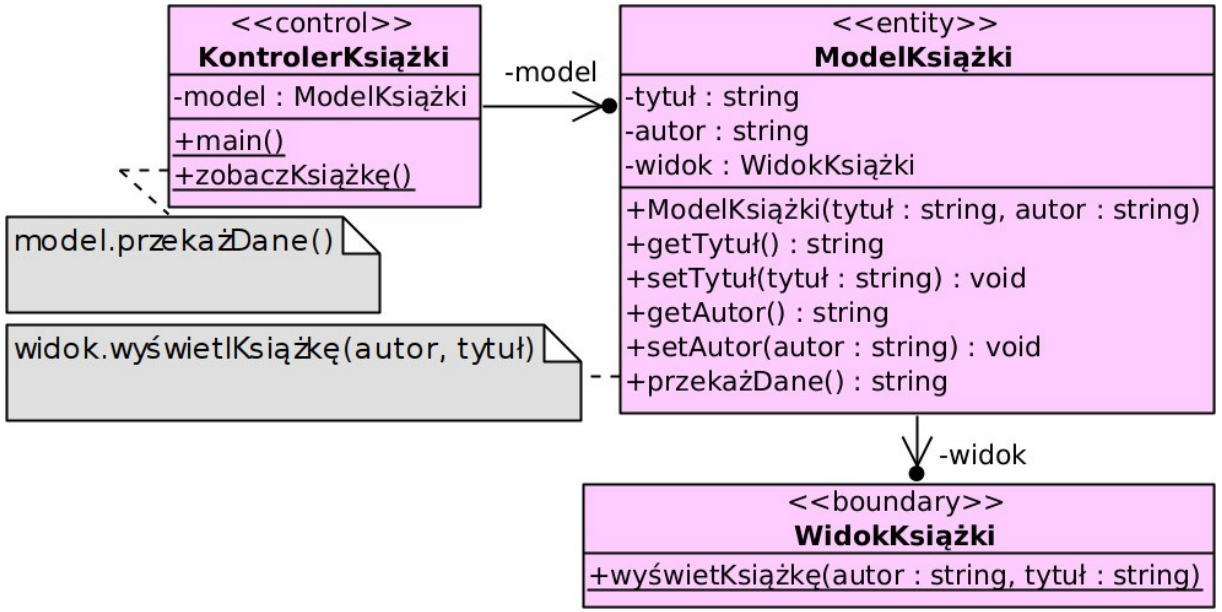
Widok

- Prezentuje dane z modelu użytkownikowi.
- Definiuje dostęp do tych danych (ich pobieranie i modyfikację).
- NIE definiuje znaczenia ani przeznaczenia tych danych.



Przykład zastosowania wzorca MVC

- Każda warstwa składa się z 1 klasy.
- Kontroler: klasa **KontrolerKsiążki** udostępnia użytkownikowi operację *zobaczKsiążkę()* z kodem *model.przekażDane()*.
- Model: obiekt **model** klasy **ModelKsiążki** wykonuje operację *przekażDane()* z kodem *widok.wyświetlKsiążkę()*.
- Widok: obiekt **widok** klasy **WidokKsiążki** wykonuje operację *wyświetlKsiążkę()*.

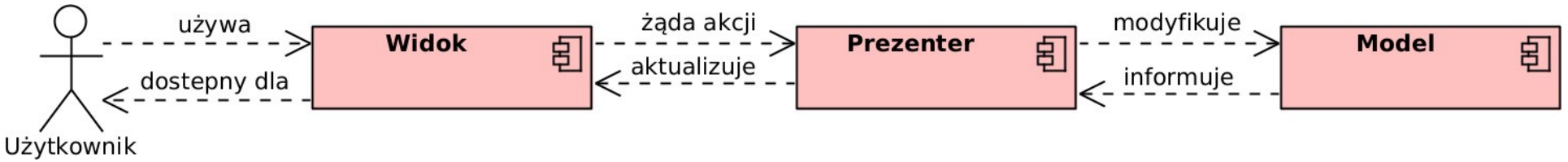


5

Wzorzec architektury MVP

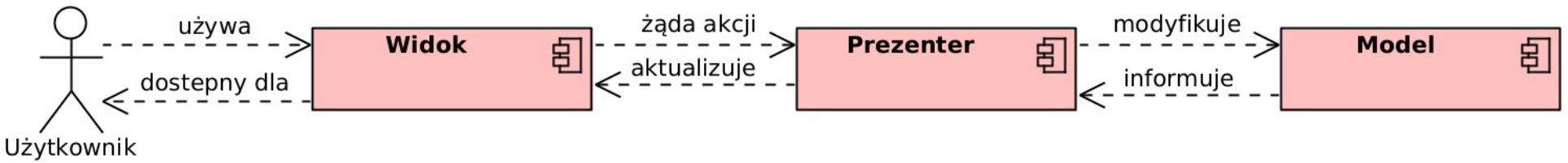
MVP (Model – View – Presenter)

- 3 warstwy oprogramowania:
  - **model** /model/ – warstwa modelu,
  - **widok** /view/ – warstwa prezentacji,
  - **prezenter** /presenter/ – warstwa biznesowa.
- Idea działania:
  - Użytkownik używa Widoku,
  - Widok od Prezentera żąda akcji odpowiedniej do działania Użytkownika,
  - Prezenter modyfikuje Model,
  - Model informuje Prezentera o zmianie swego stanu,
  - Prezenter aktualizuje Widok,
  - Widok jest dostępny dla Użytkownika.



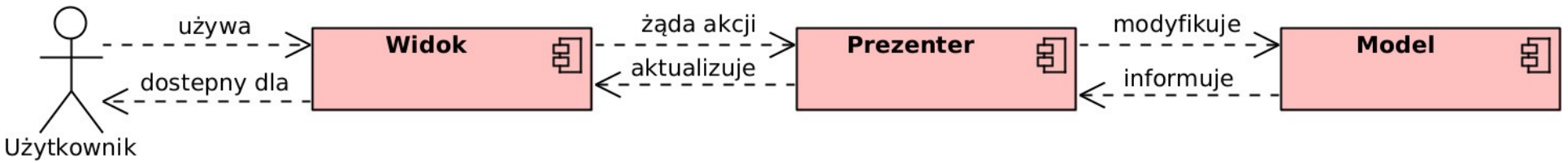
Prezenter

- Steruje przepływem danych między widokiem a modelem.
- Pobiera i przetwarza dane od widoku i od modelu.
- Definiuje dostęp do tych danych (ich pobieranie i modyfikację).
- Reaguje na zdarzenia w widoku.
- Steruje (bezpośrednio) aktualizacją widoku.



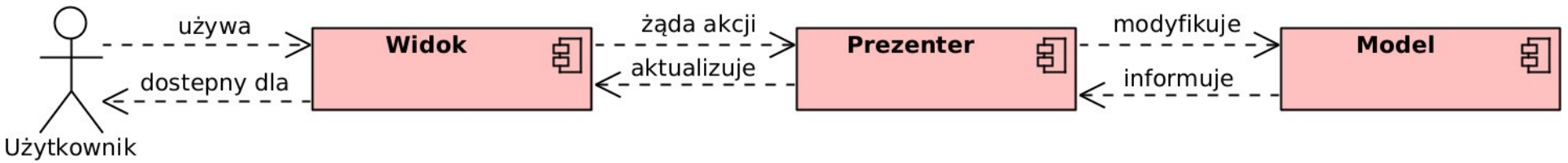
Model

- Definiuje obiekty przechowujące dane i operacje ich przetwarzania.
- Może być pasywny (obiekt sam nie zmienia swego stanu).
- Może być aktywny (obiekt sam może zmienić swój stan, niezależnie od prezentera).
- NIE definiuje sposobu prezentacji danych użytkownikowi.



Widok

- Prezentuje dane od prezentera (pośrednio z modelu) użytkownikowi.
- NIE definiuje znaczenia ani przeznaczenia tych danych.
- Reaguje na działania użytkownika
  (np. wywołuje operację biznesową prezentera).



6

Inne warstwowe wzorce architektury

Inne warstwowe wzorce architektury

PAC (Presentation – Abstraction – Control)

- Wersja wzorca MVP.
- Abstrakcja /abstraction/ – model pobierający i przetwarzający dane.
- Wiele kontrolerów (hierarchiczne drzewo).
  - Każdy kontroler zarządza swoją częścią prezentera i abstrakcji.

HMVC (Hierarchical Model – View – Controller)

- Wersja wzorca MVC.
- Analogiczna do wzorca PAC.

Inne warstwowe wzorce architektury

MVVM (Model-View-Viewmodel)

- Wersja wzorca MVC.
- Widoko-model /viewmodel/ – udostępnia i wstępnie przetwarza dane z warstwy modelu dla warstwy widoku.
- Binder – synchronizuje widok z widoko-modelem (np. w języku znaczników).

MVA (Model-View-Adapter)

- Wersja wzorca MVC.
- Warstwy modelu i widoku nie komunikują się między sobą bezpośrednio.
- Adapter – łączy konkretny model z konkretnym widokiem.
- Widok jest pasywny.

7

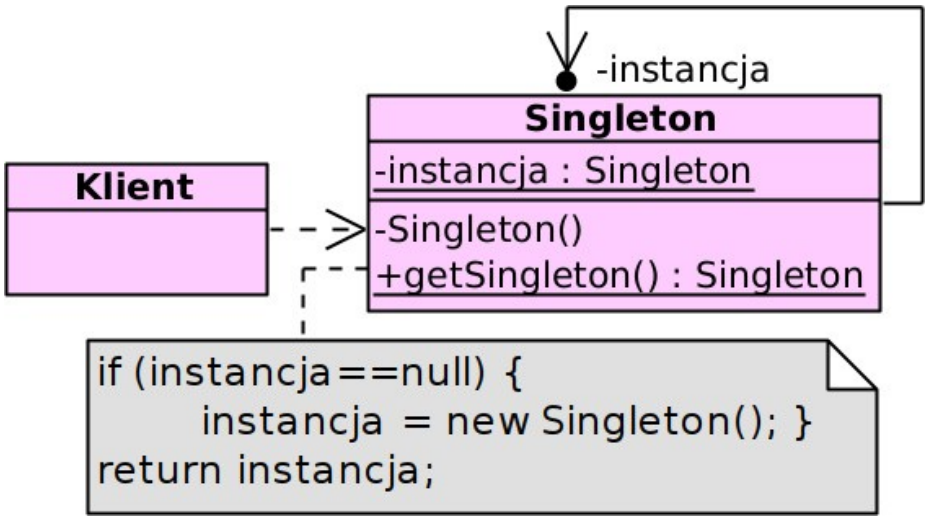
Wzorce projektowe kreacyjne

Wzorzec kreacyjny /creational/

- Opisuje sposób tworzenia, inicjacji i konfiguracji klas i obiektów (ułatwia ponowne użycie kodu).
  - Ułatwia ich modyfikację.
  - Ułatwia ponowne użycie kodu.
- **Przykłady:**
  - singleton /singleton/,
  - prototyp /prototype/,
  - metoda wytwórcza /factory method/,
  - fabryka abstrakcyjna /abstract factory/,
  - budowniczy /builder/.

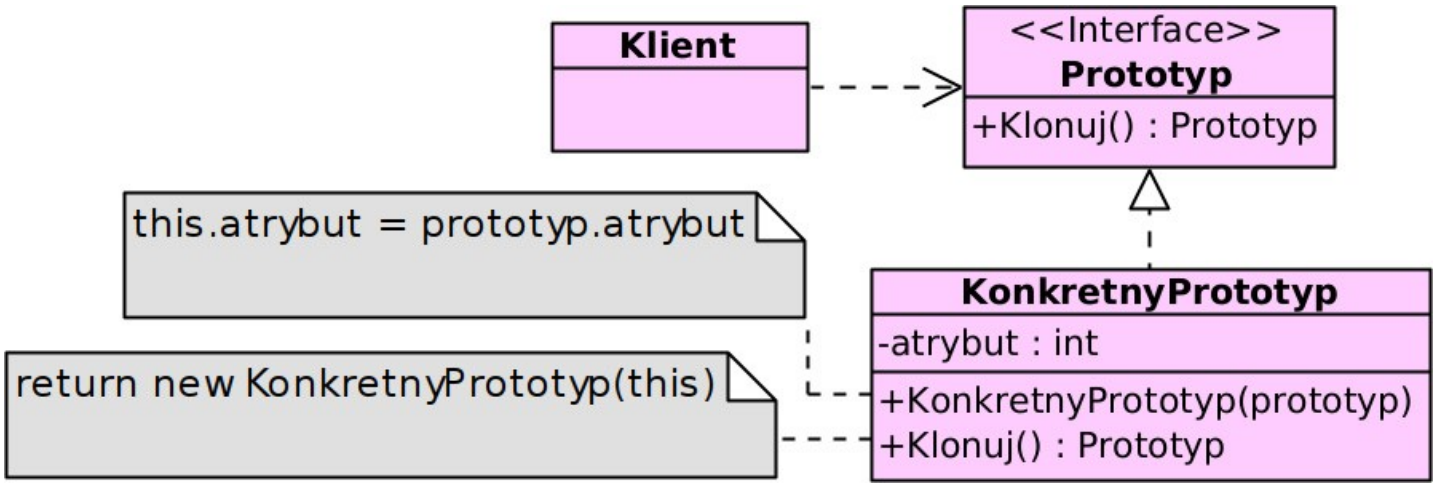
Singleton

- **Przeznaczenie:** stworzenie singletonu – klasy o globalnym dostępnie, która może mieć tylko jedną instancję.
- **Implementacja:**
  - Konstruktor singletonu jest prywatny lub chroniony i używany tylko w pierwszym wykonaniu metody go zwracającej.
  - Instancja singletonu jest jej prywatnym, statycznym atrybutem.
  - Klient ma dostęp zawsze do tego samego singletonu.



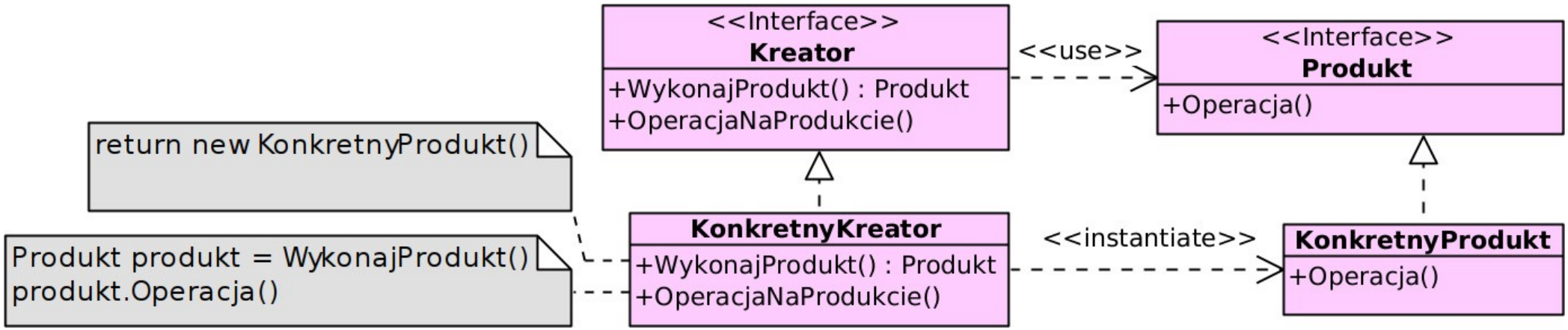
Prototyp

- **Przeznaczenie:** tworzenie prototypów – obiektów, które mogą same się powielać razem z prywatnymi atrybutami.
- **Implementacja:**
  - Interfejs prototypu zawiera metodę klonującą.
  - Prototyp go implementujący tworzy i zwraca tą metodą swoją kopię.
  - Klient pobiera klon prototypu (sam go nie tworzy).
  - Zamiast interfejsu można odpowiednio użyć abstrakcyjnej klasy.



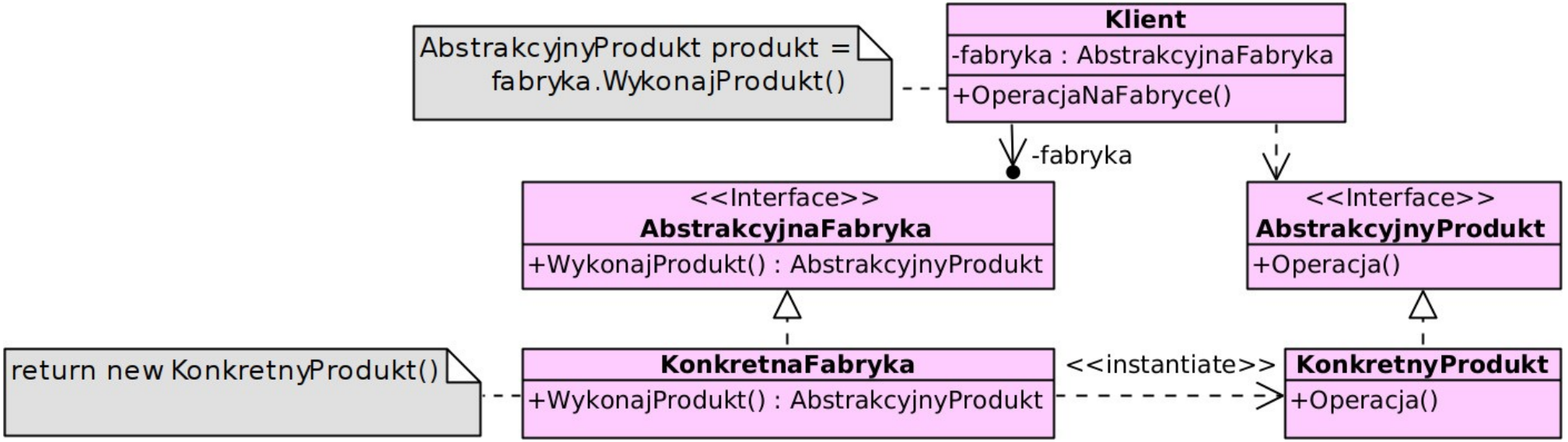
Metoda wytwórcza

- **Przeznaczenie:** tworzenie obiektu klasy bazowej o typie jej podklasy.
- **Implementacja:**
  - Interfejsy definiują kreatorów produktów i produkty.
  - Interfejs kreatora zawiera metodę tworzącą produkt.
  - Konkretny kreator tworzy konkretny produkt.
  - Zamiast interfejsów można odpowiednio użyć abstrakcyjne klasy.



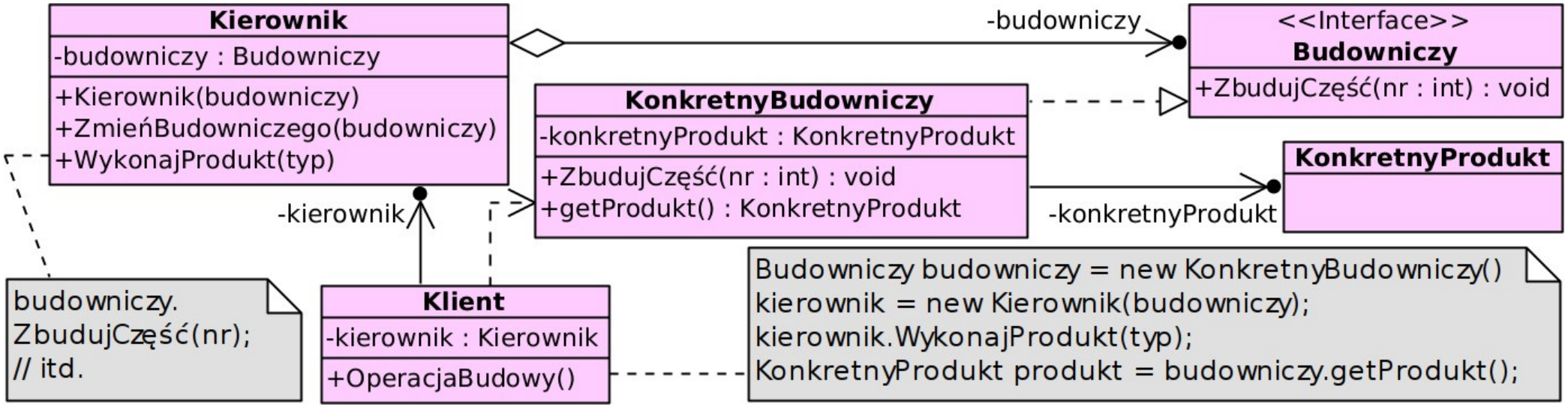
Fabryka abstrakcyjna

- **Przeznaczenie:** tworzenie rodzin spokrewnionych ze sobą lub zależnych od siebie obiektów, bez podawania ich klas.
- **Implementacja:**
  - Interfejsy definiują fabryki produktów i produkty.
  - Interfejs abstrakcyjnej fabryki zawiera metodę tworzącą produkt.
  - Konkretna fabryka tworzy konkretne produkty (tego samego rodzaju).
  - Klient konkretne produkty pobiera z konkretnej fabryki.
  - Zamiast interfejsów można odpowiednio użyć abstrakcyjne klasy.



Budowniczy

- **Przeznaczenie:** tworzenie złożonych obiektów na różne sposoby.
- **Implementacja:**
  - Interfejs budowniczego zawiera metody tworzące dane części produktu.
  - Konkretny budowniczy tworzy części konkretnego produktu i zwraca go.
  - Kierownik zawiera metodę kierującą budową produktu przez wybór budowniczego (wg decyzji klienta).
  - Klient konkretny produkt pobiera od konkretnego budowniczego.
  - Zamiast interfejsu można odpowiednio użyć abstrakcyjnej klasy.



8

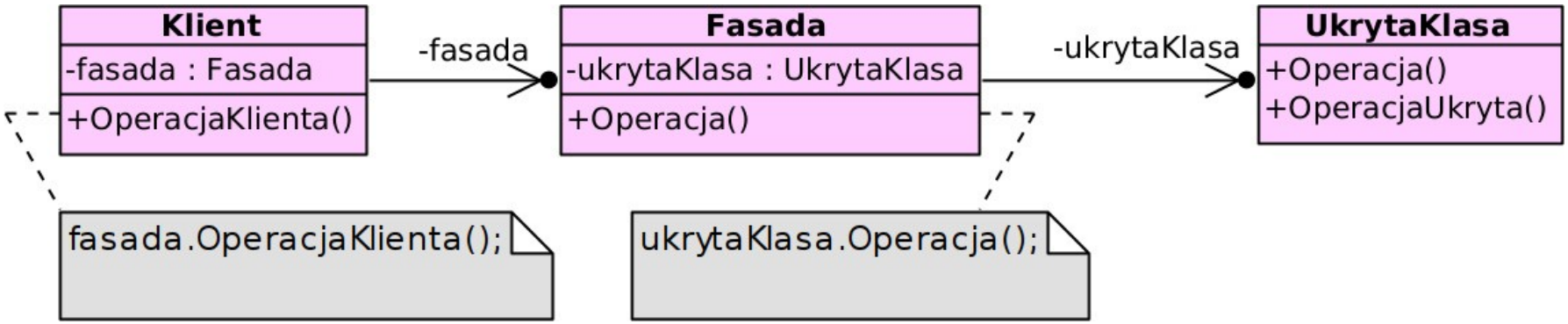
Wzorce projektowe strukturalne

Wzorzec strukturalny /structural/

- Opisuje struktury złożone z powiązanych ze sobą klas i obiektów.
  - Pokazuje sposób ich wiązania.
  - Zapewnia ich elastyczność i efektywność.
- **Przykłady:**
  - **fasada** /facade/,
  - **adapter** /adapter/,
  - **dekorator** /decorator/,
  - **kompozyt** /composite/,
  - **pełnomocnik** /proxy/,
  - **pyłek** /flyweight/,
  - **most** /bridge/.

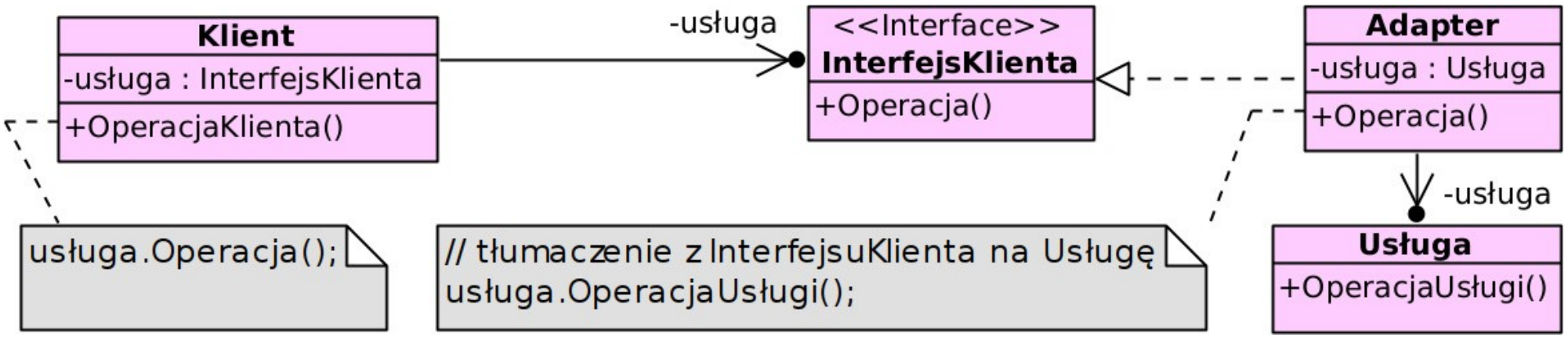
Fasada

- **Przeznaczenie:** dostęp do układu klas / obiektów przez jeden – fasadę.
- **Implementacja:**
  - Fasada zawiera referencje do klas, których metody ma udostępniać.
  - Fasada udostępnia tylko potrzebne metody tych klas.
  - Te klasy i ich obiekty nie są widoczne dla klienta.
  - Klient ma dostęp tylko do fasady.



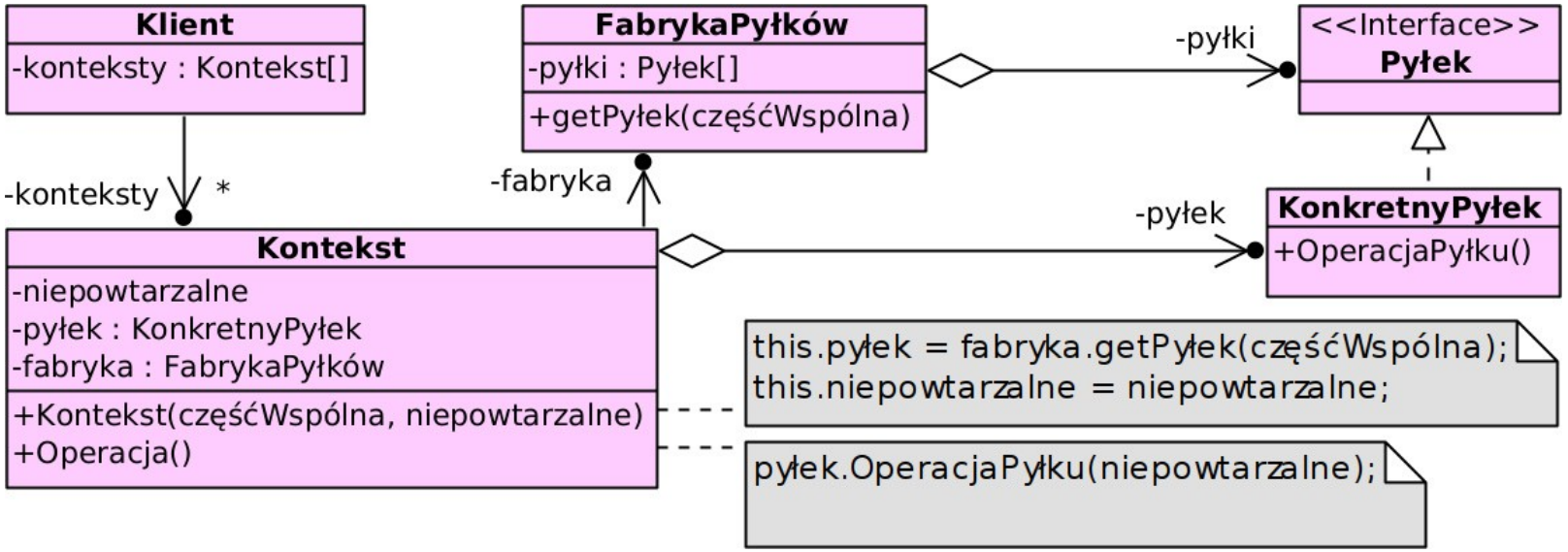
Adapter

- **Przeznaczenie:** dostęp do obiektu o nieobsługiwanym interfejsie przez interfejs pośredni – adapter.
- **Implementacja:**
  - Adapter implementuje interfejs klienta i opakowuje obiekt usługi.
  - Adapter tłumaczy „operacje” klienta na „operacje” obiektu usługi.
  - Zamiast interfejsu można odpowiednio użyć abstrakcyjnej klasy.



Pyłek

- **Przeznaczenie:** współdzielenie wspólnej części wielu podobnych obiektów tej samej klasy w jednym – w pyłku.
- **Implementacja:**
  - Powtarzalną część obiektu (operacje, dane) kontekst zapisuje w pyłku.
  - Fabryka tworzy nowy lub znajduje gotowy konkretny pyłek i zwraca go.
  - Niepowtarzalną część obiektu kontekst zapisuje w sobie.
  - Raz utworzony pyłek nie może zostać zmieniony.
  - Zamiast interfejsu można odpowiednio użyć abstrakcyjnej klasy.



Przeznaczenie pozostałych wzorców

- **Dekorator** – dynamiczne dodawanie nowych operacji do obiektów bez zmiany definicji ich klas, przez opakowanie ich innym obiektem – dekoratorem.
- **Kompozyt** – łączenie różnych obiektów w kolekcję.
- **Pełnomocnik** – zastępowanie obiektu innym obiektem – pełnomocnikiem, który kontroluje do niego dostęp.
- **Most** – uniezależnienie abstrakcji obiektu od jego konkretnej realizacji.

9

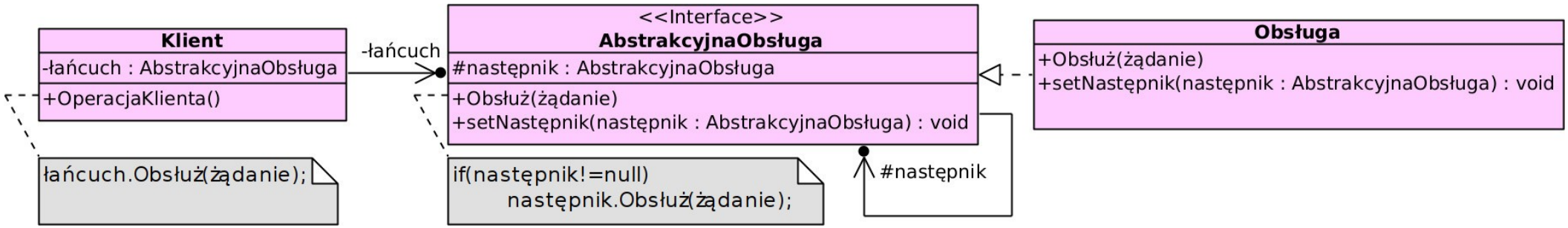
Wzorce projektowe czynnościowe

Wzorzec czynnościowy /behavioral/

- Opisuje zachowanie, interakcję i odpowiedzialność współpracujących ze sobą klas i obiektów (podział zadań).
  - Rozdziela zadania między nimi.
  - Modeluje algorytmy.
- **Przykłady:**
  - łańcuch zobowiązań /chain of responsibility/,
  - metoda szablonowa /templete method/,
  - pamiątka /memento/,
  - iterator /iterator/,
  - mediator /mediator/,
  - interpreter /interpreter/,
  - obserwator /observer/,
  - stan /state/,
  - strategia /strategy/,
  - polecenie /command/,
  - odwiedzający /visitor/.

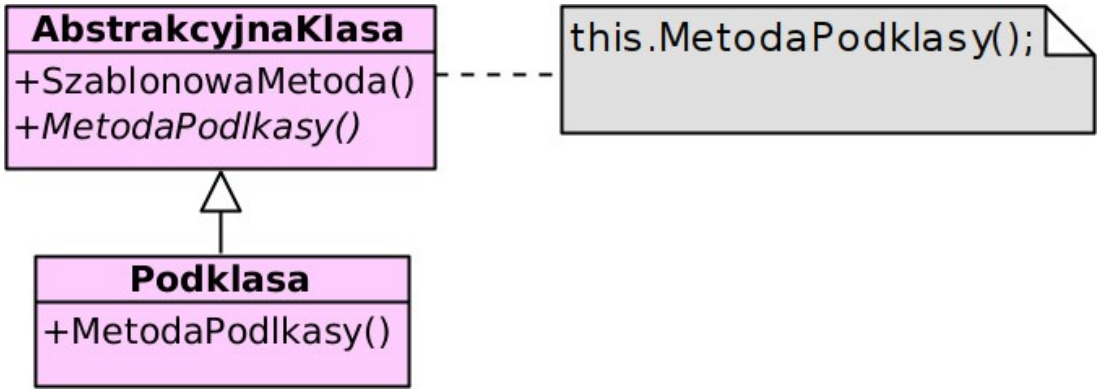
Łańcuch zobowiązań

- **Przeznaczenie:** przekazanie wykonania operacji łańcuchowi obiektów, które kolejno ją (lub jej część) wykonują, jeśli powinny.
- **Implementacja:**
  - Łańcuch jest jednokierunkową listą obiektów.
  - Interfejs definiuje te obiekty.
  - Żądanie wykonania operacji przekazywane jest kolejno od początku aż do końca łańcucha, lub aż do jej ukończenia lub odrzucenia.
  - Zamiast interfejsu można odpowiednio użyć abstrakcyjnej klasy.



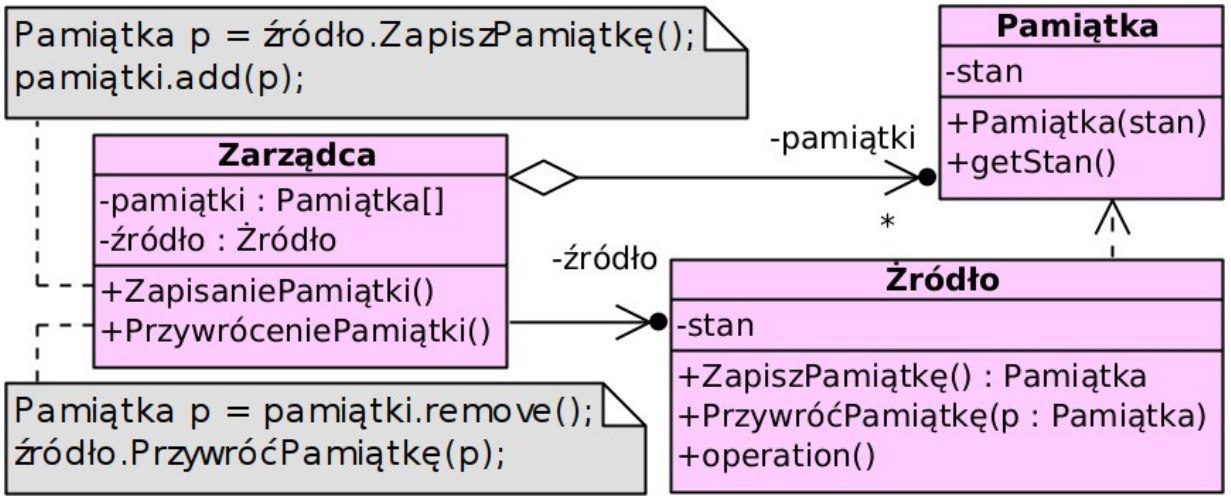
Metoda szablonowa

- **Przeznaczenie:** definiowanie w klasie bazowej metody szablonowej – szablonu algorytmu dla jej podklas.
- **Implementacja:**
  - Klasa abstrakcyjna definiuje i implementuje metody szablonowe, wspólne dla jej podklas.
  - Klasa abstrakcyjna definiuje abstrakcyjne metody dla jej podklas.
  - Podklasy po niej dziedziczące implementują abstrakcyjne metody własnym kodem.



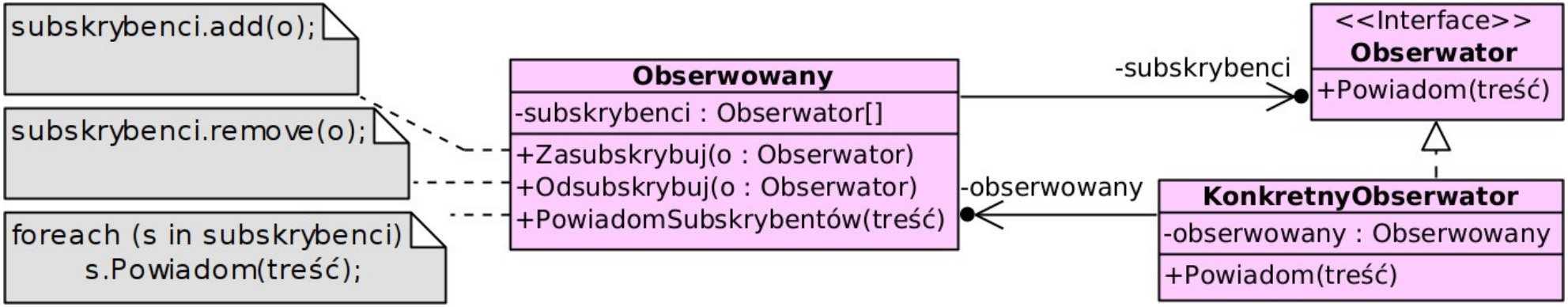
Pamiątka

- **Przeznaczenie:** zapisanie / przywrócenie stanu obiektu (źródła) do / z obiektu pamiątki razem z jego prywatną zawartością.
- **Implementacja:**
  - Źródło zawiera operację tworzenia pamiątki ze swoim stanem.
  - Źródło zawiera operację przywracania swojego stanu z pamiątki.
  - Pamiątka jest niezmienna.
  - Zarządca tworzy i przywraca pamiątkę pośrednio przez źródło.
  - Źródło ma pełny dostęp do swojej pamiątki; zarządca ma dostęp tylko do jej metadanych (nie do stanu); inne obiekty nie mają dostępu.



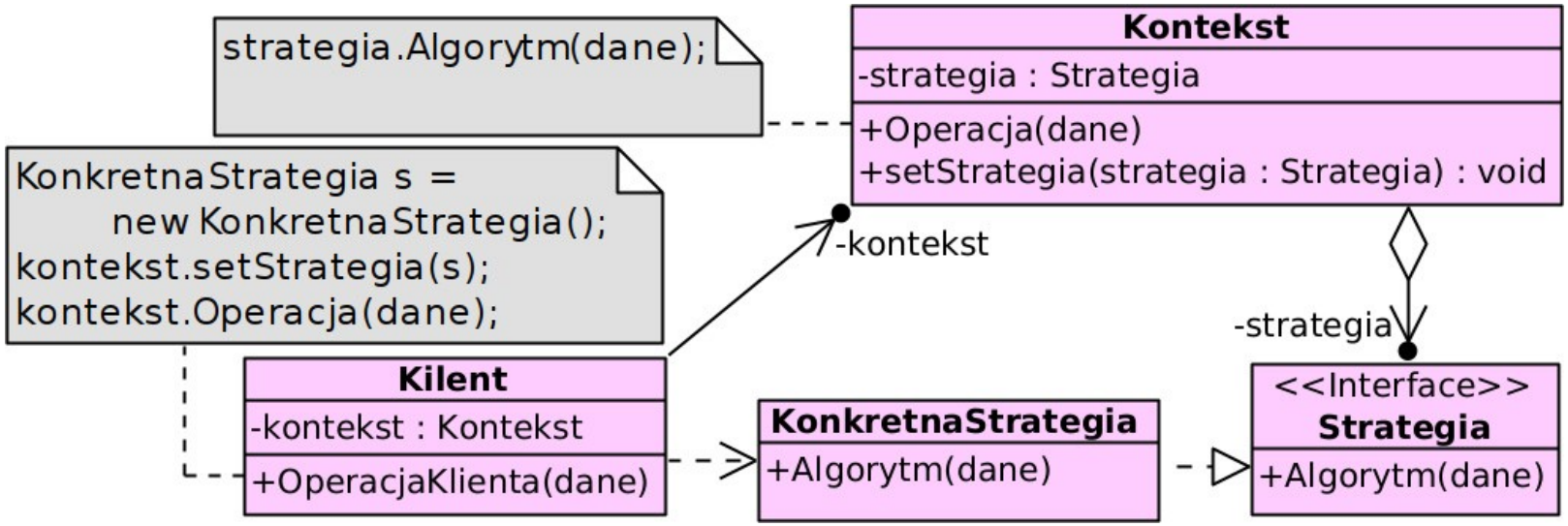
Obserwator

- **Przeznaczenie:** obserwowanie obiektów
  i o zdarzeniach w nich zachodzących informowanie innych obiektów.
- **Implementacja:**
  - Obserwowany obiekt zawiera listę obserwujących go obserwatorów (subskrybentów).
  - Obserwowany zawiera publiczne metody subskrypcji i powiadamiania obserwatorów o zmianie swojego stanu.
  - Obserwator zawiera publiczną metodę przyjmowania powiadomień.
  - Alternatywnie: zmiany stanu (zdarzenia) obserwowanego obiektu będą znane obserwatorowi, gdy będzie miał do niego referencję.
  - Zamiast interfejsu można odpowiednio użyć abstrakcyjnej klasy.



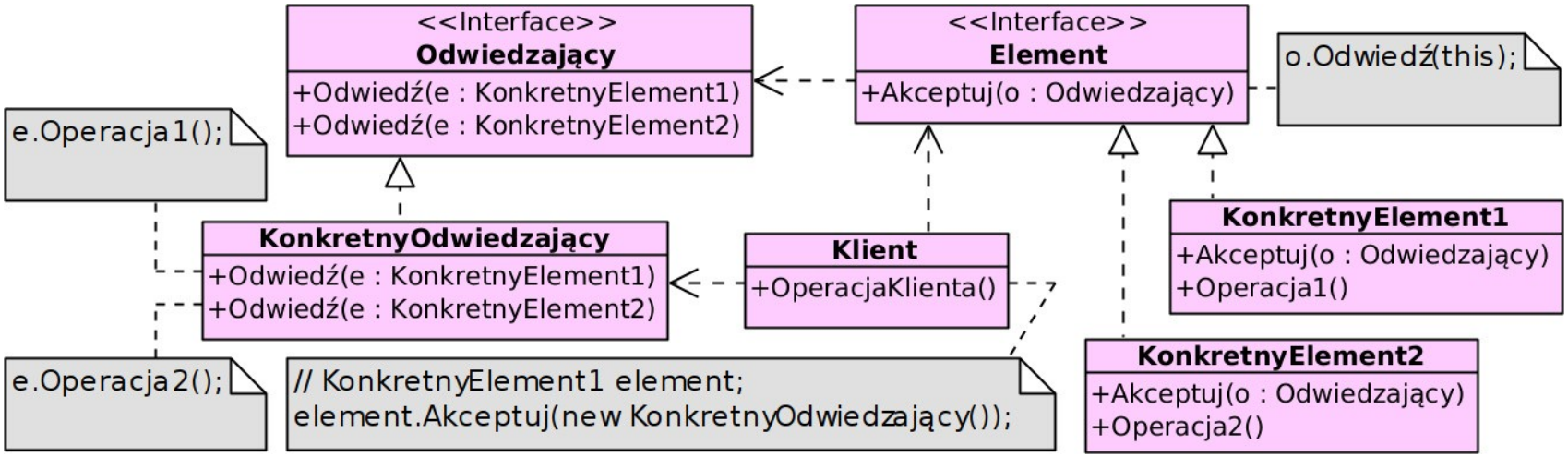
Strategia

- **Przeznaczenie:** zdefiniowanie alternatywnych operacji jako operacje alternatywnie wybieralnych obiektów – strategii.
- **Implementacja:**
  - Interfejs strategia zawiera operację dostępu do konkretnych strategii.
  - Konkretna strategia implementuje tę operację konkretnym algorytmem.
  - Obiekt kontekst zawiera referencję do wybranej konkretnej strategii.
  - Kontekst wywołuje jej operację, nie znając jej implementacji.
  - Zamiast interfejsu można odpowiednio użyć abstrakcyjnej klasy.



Odwiedzający

- **Przeznaczenie:** oddzielenie algorytmu operacji od obiektu, którego dotyczą (odwiedzanego) i umieszczenie jej w innej klasie (odwiedzającym).
- **Implementacja:**
  - Interfejs odwiedzającego zawiera wirtualne operacje, przeznaczone każda do odwiedzenia obiektów innej klasy (każdy odwiedzający implementuje je, znając definicję klas odwiedzanych obiektów).
  - Odwiedzane obiekty implementują interfejs element z operacją akceptacji – przyjęcia odwiedzającego.
  - Operacja akceptacji przekazuje wywołujący ją obiekt do odpowiedniej operacji odwiedzającego, którego jest ta wywoływana akceptacja.
  - Zamiast interfejsów można odpowiednio użyć abstrakcyjne klasy.



Przeznaczenie pozostałych wzorców

- **Iterator** – sekwencyjne operowanie na elementach różnie zdefiniowanych kolekcji.
- **Mediator** – wykonywanie operacji komunikacji między powiązanymi obiektami przez pośredniczący obiekt – mediator, a nie bezpośrednio.
- **Interpreter** – opisanie gramatyki języka i interpretowanie zdania według niej.
- **Stan** – zmiana zachowania (wykonywanych operacji) obiektu, gdy obiekt zmienia swój stan.
- **Polecenie** – zdefiniowanie różnych żądań jako różne obiekty, aby móc je opóźniać, kolejkować i parametryzować.

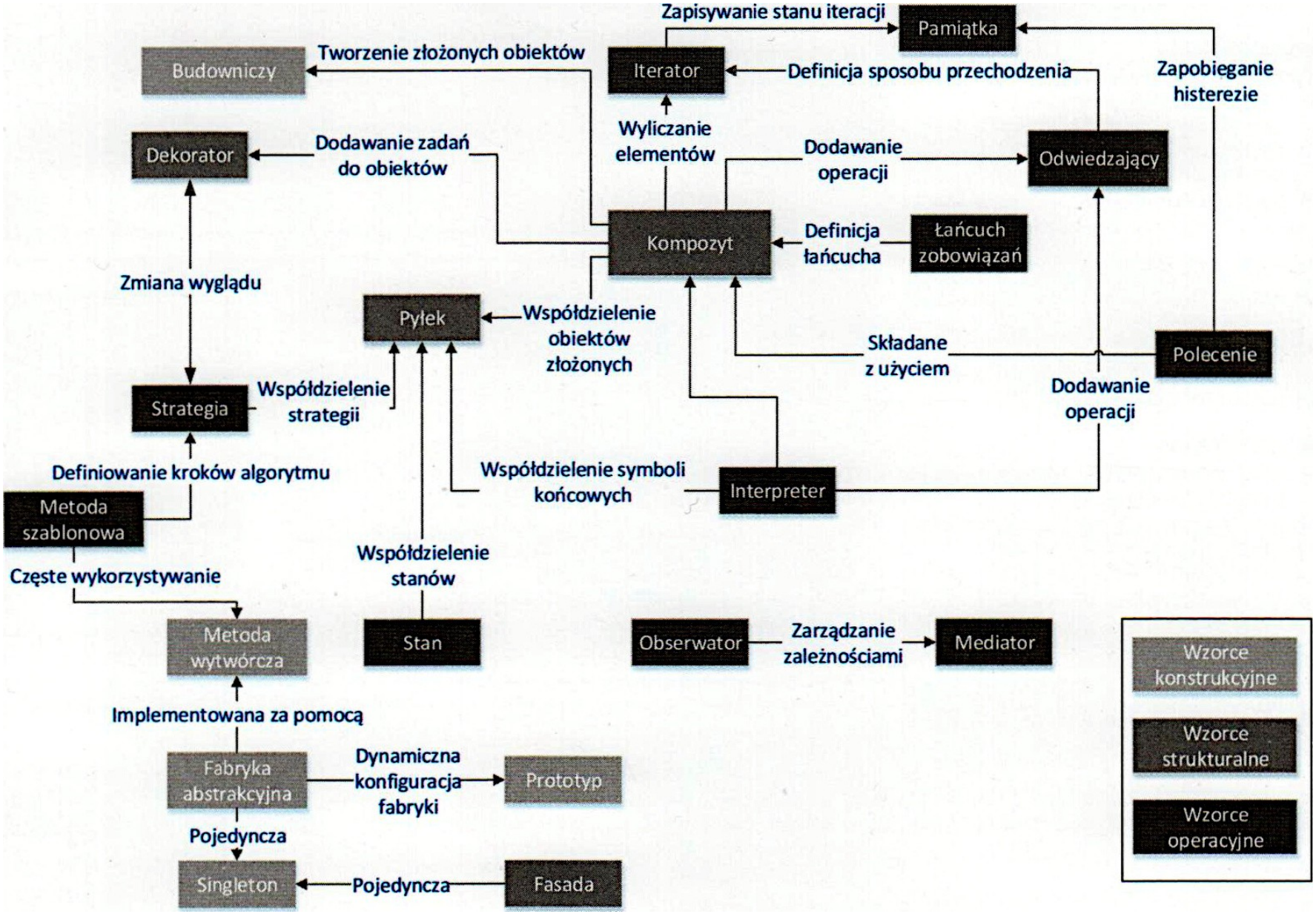
10

Relacje między wzorcami projektowymi

Relacje między wzorcami projektowymi

Wzorce projektowe

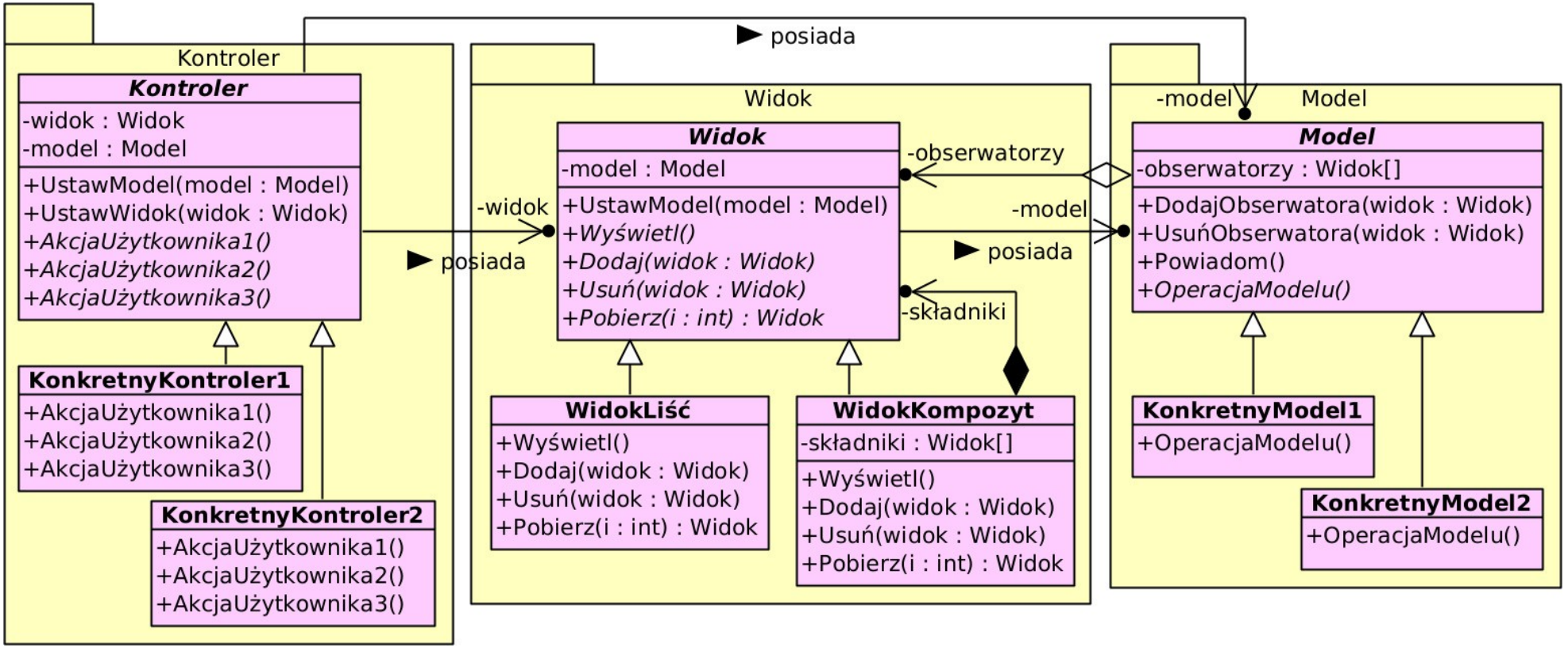
źródło: „Tablice informatyczne. Wzorce projektowe”, Daniel Krasnokucki, wyd. Helion 2017



Przykład wzorca MVC z wzorcami projektowymi

Na podst. „Bureaucracy: Pattern languages of program design 3”; wyd. Addison-Weseley, 1997

- **Strategia** w kontrolerze i widoku – wybór konkretnego kontrolera.
- **Obserwator** w widoku i modelu – powiadamianie widoku przez model o zmianie swego stanu.
- **Kompozyt** w widoku – praca z zagnieżdżonymi widokami.



11

Złożone wzorce projektowe

Przykłady

- TODO Te od refaktoryzacji z prezentacji ZK.

Temat następnej prezentacji

Definiowanie metamodelu