Zaawansowane programowanie obiektowe - wykład 4

dr Piotr Jastrzębski

Wzorce kreacyjne - cd.

Wzorce kreacyjne

- Budowniczy
- Fabryka abstrakcyjna
- Metoda wytwórcza
- Prototyp
- Singleton

Fabryka abstrakcyjna

Przeanalizujmy przykład: http://tomaszjarzynski.pl/fabryka-abstrakcyjna-wzorzec-projektowy-abstract-factory/

Zastosowania:

- komunikacja z różnymi api
- wyświetlanie tekstu z różnych formatów plików
- obsługa różnych typów baz danych

Metoda wytwórcza

Przykład do przeanalizowania: http://tomaszjarzynski.pl/metoda-wytworcza-wzorzec-projektowy-factory-method/

Zastosowania:

- ukrycie przed klientem implementacji
- sterowniki

Prototyp

Prototyp – kreacyjny wzorzec projektowy, którego celem jest umożliwienie tworzenia obiektów danej klasy bądź klas z wykorzystaniem już istniejącego obiektu, zwanego prototypem.

Przykład do analizy: https://pl.wikibooks.org/wiki/Kody_%C5%BAr%C3%B3d%C5%82owe/Prototyp_(wzorzec_projektowy)#C#

Zastosowania

- duża liczba obiektów tego samego typu
- chcemy otrzymać prawidłową kopię obiektu

Singleton

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

```
public sealed class Singleton
{
    private static Singleton instance = null;

    private Singleton()  {  }

    public static Singleton Instance()
    {
        if (instance == null)
        {
            instance = new Singleton();
        }
        return instance;
    }

}
```

Wzorce strukturalne

Wzorce strukturalne

- Adapter
- Dekorator
- Fasada
- Kompozyt
- Most
- Pełnomocnik
- Pyłek

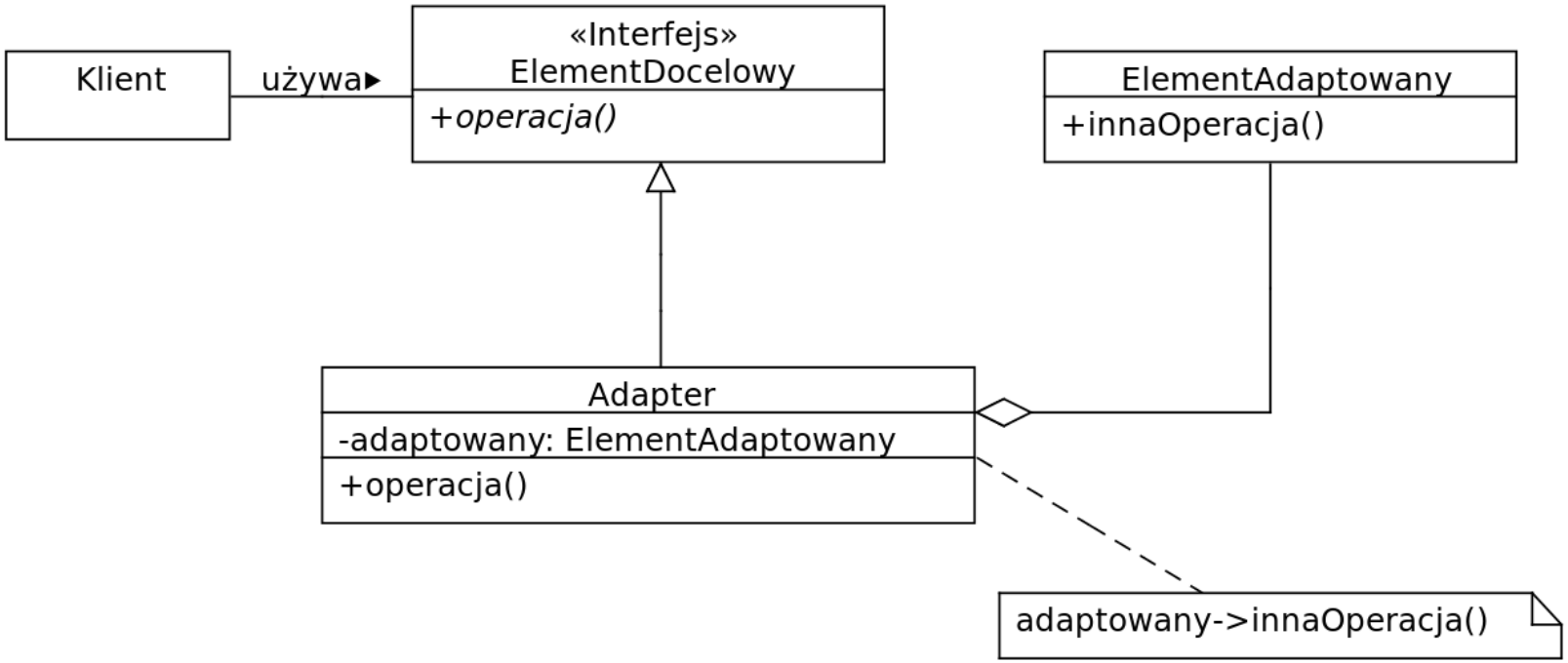
Wzorce strukturalne opisują struktury powiązanych obiektów oraz sposoby składania obiektów w większe struktury.

Adapter

Adapter (także: opakowanie, ang. wrapper) – strukturalny wzorzec projektowy, którego celem jest umożliwienie współpracy dwóm klasom o niekompatybilnych interfejsach. Adapter przekształca interfejs jednej z klas na interfejs drugiej klasy.

Przykład:

- https://github.com/pjastr/ZPO_programy/tree/master/AdapterWzorzecProjektowy

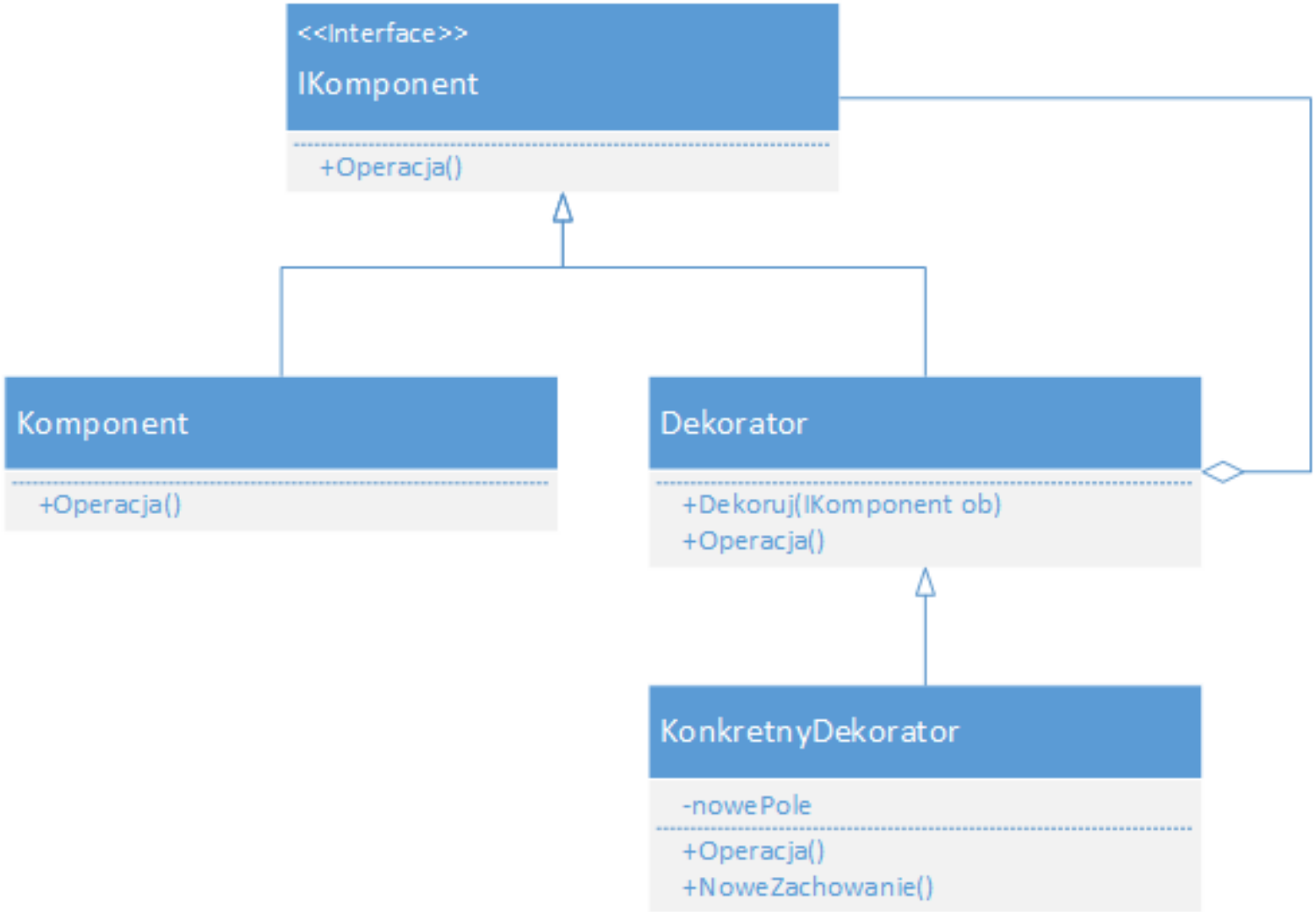

Klient
używa
«Interfejs»
ElementDocelowy
+operacja()
ElementAdaptowany
+innaOperacja()
Adapter
-adaptowany: ElementAdaptowany
+operacja()
adaptowany->innaOperacja()

Rysunek 1:

Dekorator

Dekorator – wzorzec projektowy należący do grupy wzorców strukturalnych. Pozwala na dodanie nowej funkcji do istniejących klas dynamicznie podczas działania programu. Dekoratory są alternatywą dla dziedziczenia. Dziedziczenie rozszerza zachowanie klasy w trakcie kompilacji, w przeciwieństwie do dekoratorów, które rozszerzają klasy w czasie działania programu.

Przykład:
https://github.com/pjastr/ZPO_programy/tree/master/Dekorator


<<Interface>>
IKomponent
+Operacja()
Komponent
+Operacja()
Dekorator
+Dekoruj(IKomponent ob)
+Operacja()
KonkretnyDekorator
-nowePole
+Operacja()
+NoweZachowanie()

Rysunek 2:

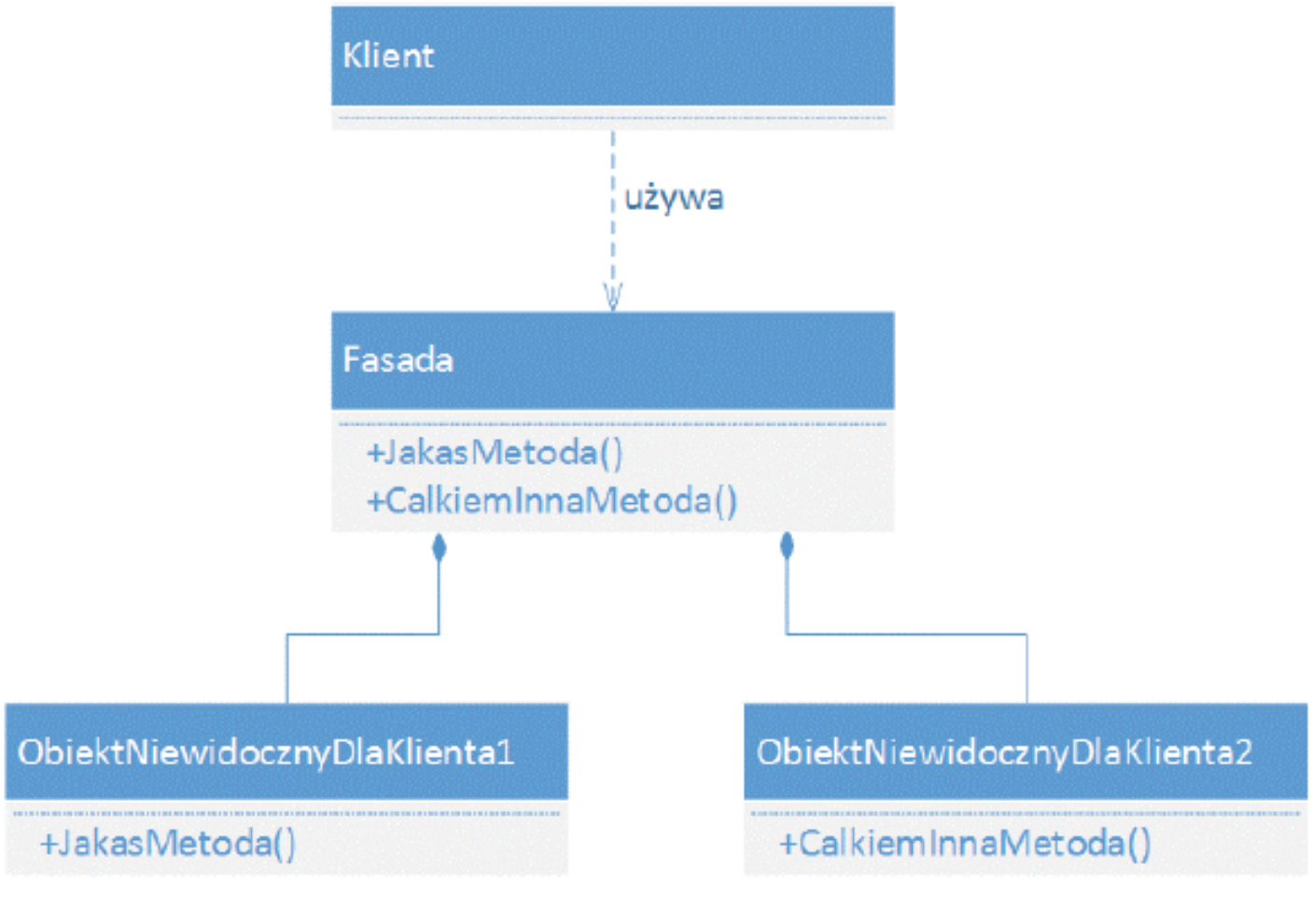
Zastosowania:

- dynamiczna zmiana wyglądu okna w zależności od potrzeb,
- dodawanie funkcji do istniejących klas w czasie działania programu.

Fasada

Fasada – wzorzec projektowy należący do grupy wzorców strukturalnych. Służy do ujednolicenia dostępu do złożonego systemu poprzez wystawienie uproszczonego, uporządkowanego interfejsu programistycznego, który ułatwia jego użycie.

Przykład:
https://github.com/pjastr/ZPO_programy/tree/master/Fasada


Klient
używa
Fasada
+JakasMetoda()
+CalkiemInnaMetoda()
ObiektNiewidocznyDlaKlienta1
+JakasMetoda()
ObiektNiewidocznyDlaKlienta2
+CalkiemInnaMetoda()

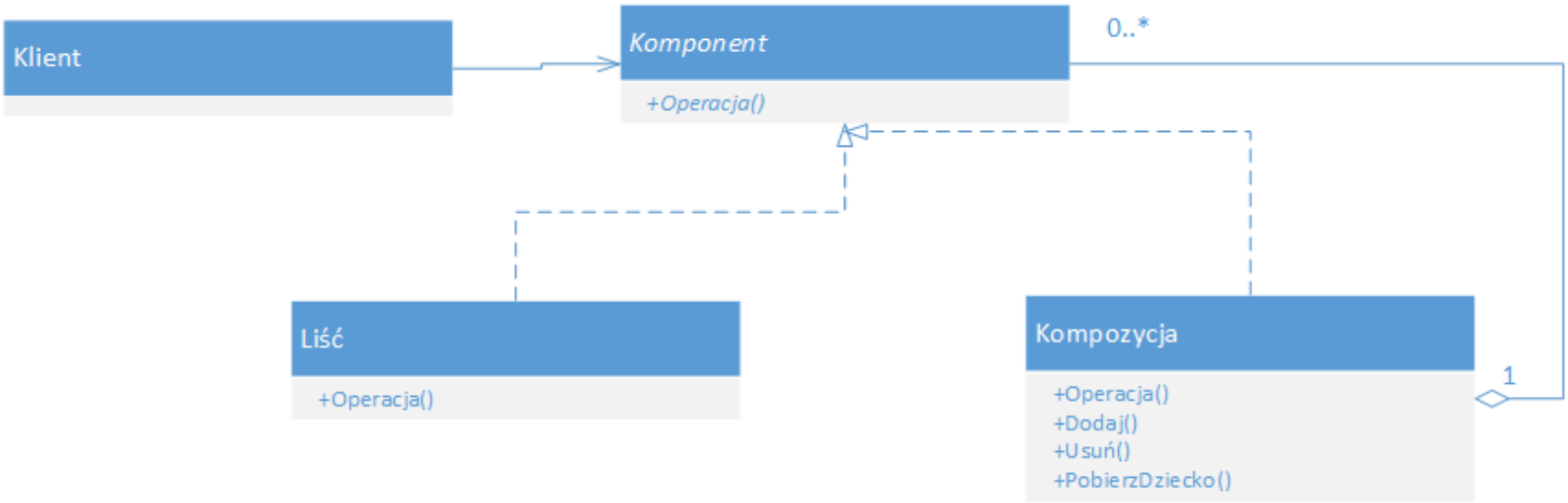
Rysunek 3:

- ukrycie części systemu przed klientem (np. w banku, sklepie) i zmniejszenie liczby zależności pomiędzy klientem a systemem,
- API do połączenia z naszą aplikacją.

Kompozyt

Kompozyt – strukturalny wzorzec projektowy, którego celem jest składanie obiektów w taki sposób, aby klient widział wiele z nich jako pojedynczy obiekt.

Przykład: https://github.com/pjastr/ZPO_programy/tree/master/Kompozyt


Klient
Komponent
+Operacja()
0..*
Liść
+Operacja()
Kompozycja
+Operacja()
+Dodaj()
+Usuń()
+PobierzDziecko()
1

Rysunek 4:

- Liść (Leaf) – reprezentuje prymitywny obiekt nie posiadający potomków,
- Kompozyt (Composite) – reprezentuje grupę obiektów, składającą się z „liści”, implementuje akcje interfejsu Komponent,
- Komponent (Component) – interfejs, który implementują obiekty, definiuje ich domyślne zachowanie,
- Klient – operuje na obiektach zawartych w układzie.

Zastosowania

- kiedy potrzebujemy systemu z możliwością łatwego rozszerzania o nowe komponenty,
- np. sprzedaż produktów, zestawów produktów, usług.

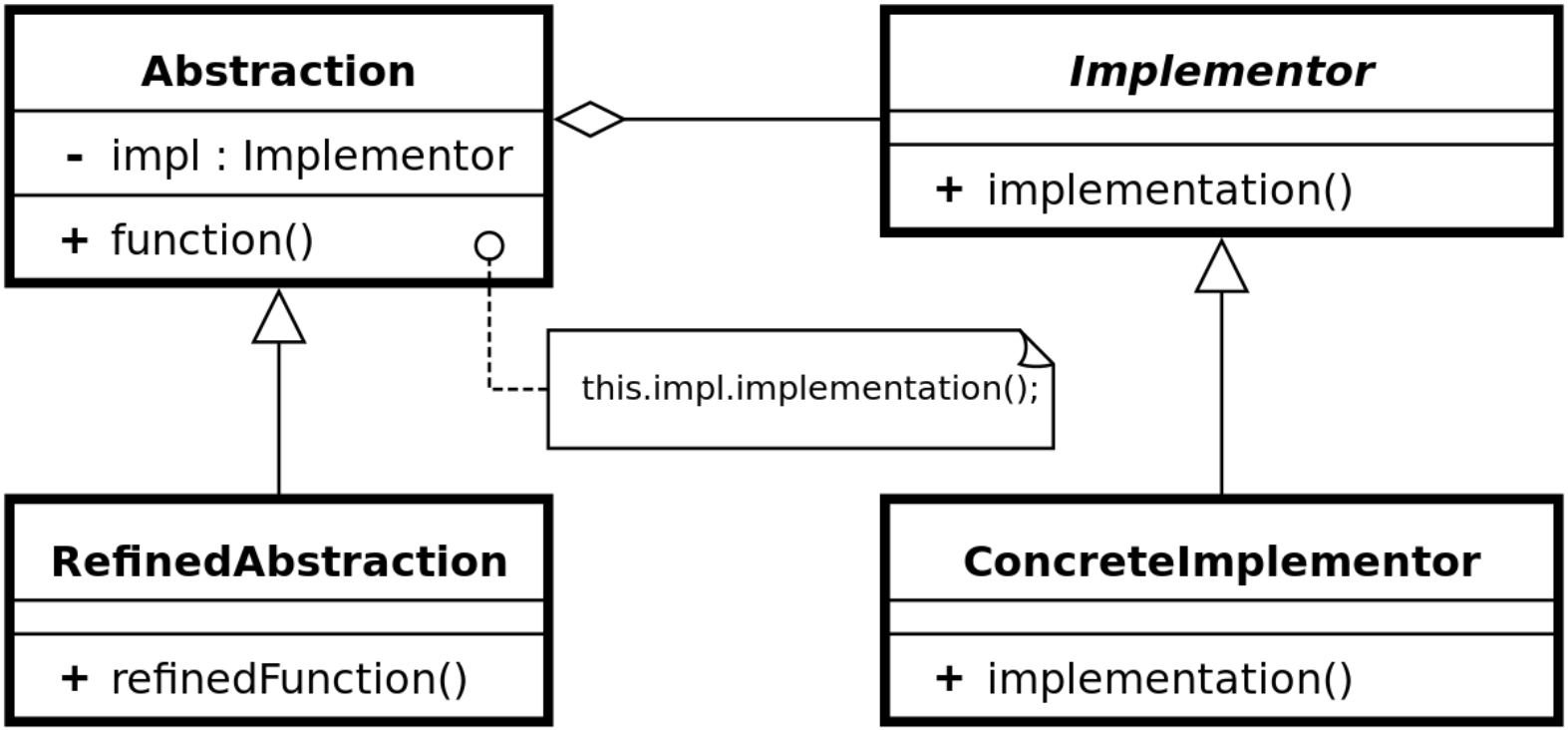
Most

Wzorzec mostu (ang. Bridge pattern) – strukturalny wzorzec projektowy, który 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.

Przykład: https://github.com/pjastr/ZPO_programy/tree/master/MostWzorzec


Abstraction
- impl : Implementor
+ function()
Implementor
+ implementation()
this.impl.implementation();
RefinedAbstraction
+ refinedFunction()
ConcreteImplementor
+ implementation()

Rysunek 5:

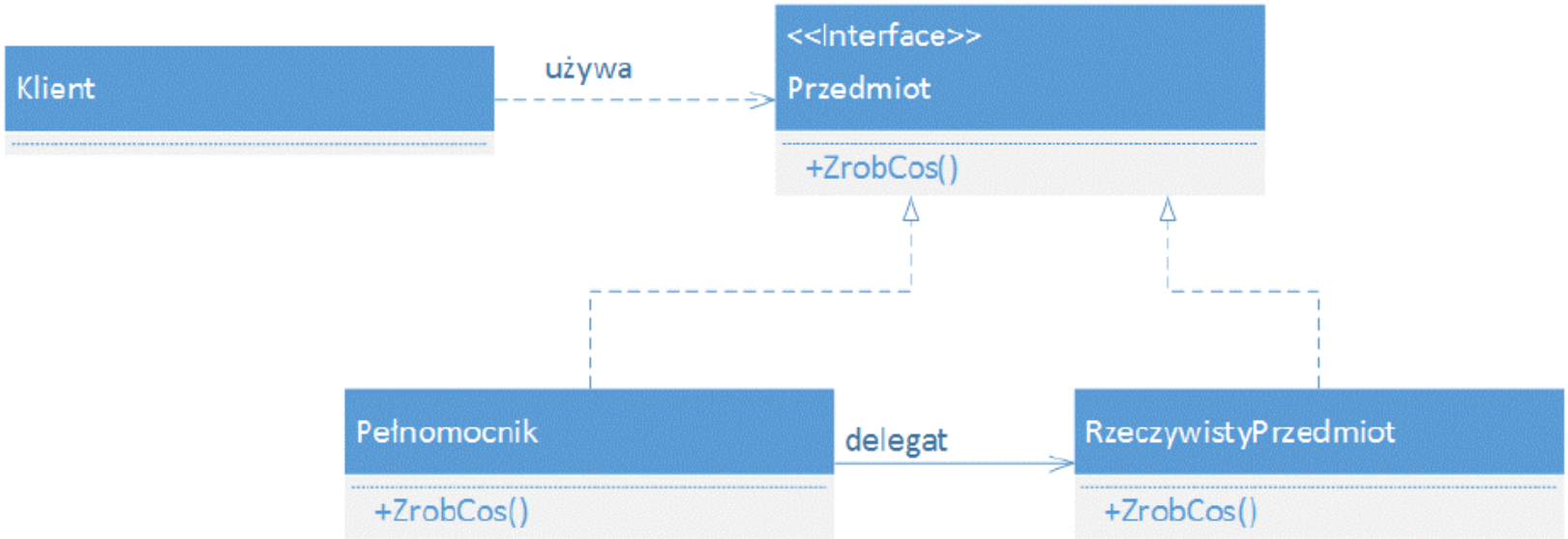
Pełnomocnik

Pełnomocnik (ang. proxy) – strukturalny wzorzec projektowy, którego celem jest utworzenie obiektu zastępującego inny obiekt. Stosowany jest w celu kontrolowanego tworzenia na żądanie kosztownych obiektów oraz kontroli dostępu do nich.

Przykład: https://github.com/pjastr/ZPO_programy/tree/master/PelnomocnikWzorzec

Rodzaje:

- wirtualny – przechowuje obiekty, których utworzenie jest kosztowne; tworzy je na żądanie
- ochraniający – kontroluje dostęp do obiektu sprawdzając, czy obiekt wywołujący ma odpowiednie prawa do obiektu wywoływanego
- zdalny – czasami nazywany ambasadorem; reprezentuje obiekty znajdujące się w innej przestrzeni adresowej
- sprytne odwołanie – czasami nazywany sprytnym wskaźnikiem; pozwala na wykonanie dodatkowych akcji podczas dostępu do obiektu, takich jak: zliczanie referencji do obiektu czy ładowanie obiektu do pamięci

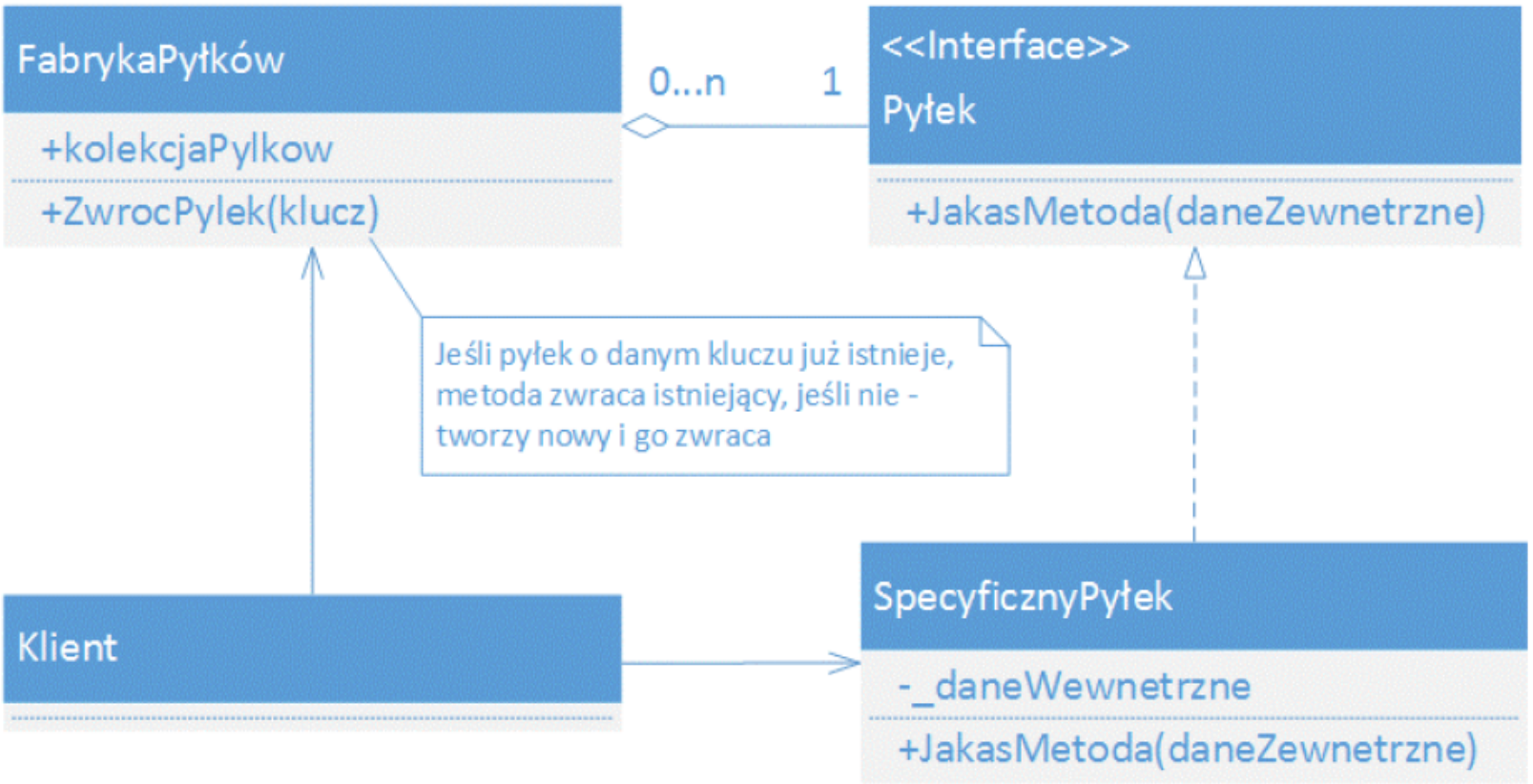

Klient
używa
<<Interface>>
Przedmiot
+ZrobCos()
Pełnomocnik
+ZrobCos()
delegat
RzeczywistyPrzedmiot
+ZrobCos()

Rysunek 6:

Pyłek

Pyłek (ang. Flyweight) – strukturalny wzorzec projektowy, którego celem jest zmniejszenie wykorzystania pamięci poprzez poprawę efektywności obsługi dużych obiektów zbudowanych z wielu mniejszych elementów poprzez współdzielenie wspólnych małych elementów.

Przykład: https://github.com/pjastr/ZPO_programy/tree/master/Py%C5%82ekWzorzec


FabrykaPyłków
+kolekcjaPylkow
+ZwrocPylek(klucz)
0...n
1
<<Interface>>
Pyłek
+JakasMetoda(daneZewnetrzne)
Jeśli pyłek o danym kluczu już istnieje, metoda zwraca istniejący, jeśli nie - tworzy nowy i go zwraca
Klient
SpecyficznyPyłek
-_daneWewnetrzne
+JakasMetoda(daneZewnetrzne)

Rysunek 7:

Bibliografia

- Daniel Krasnokucki, Wzorce projektowe. Leksykon kieszonkowy, Wyd. Helion 2017.
- http://tomaszjarzynski.pl/metoda-wytworcza-wzorzec-projektowy-factory-method/, dostęp online 1.03.2019.
- http://tomaszjarzynski.pl/fabryka-abstrakcyjna-wzorzec-projektowy-abstract-factory/
- http://www.altcontroldelete.pl/artykuly/wzorzec-adapter-przykladowa-implementacja-w-c/. dostęp online 10.03.2019.
- http://lukaszkosiorowski.pl/programowanie/dekorator-decorator/, dostęp online 10.03.2019.

- http://lukaszkosiorowski.pl/programowanie/kompozyt-composite/, dostęp online 10.03.2019.