Programowanie obiektowe

Laboratorium 11 - przegląd wybranych wzorców

mgr inż. Krzysztof Szwarc

krzysztof@szwarc.net.pl

Sosnowiec, 24 maja 2017

Wzorce projektowe

Opis

Wzorce projektowe (ang. Design Patterns) są uniwersalnymi rozwiązaniami dla często pojawiających się problemów. Wyróżniamy ich trzy rodzaje:

- Kreacyjne (opisują proces tworzenia nowych obiektów; np. singleton).
- Strukturalne (opisują struktury powiązanych ze sobą obiektów; np. fasada).
- Czynnościowe (opisują zachowanie i odpowiedzialność współpracujących obiektów; np. strategia).

Wzorce kreacyjne

Singleton - (anty)wzorzec

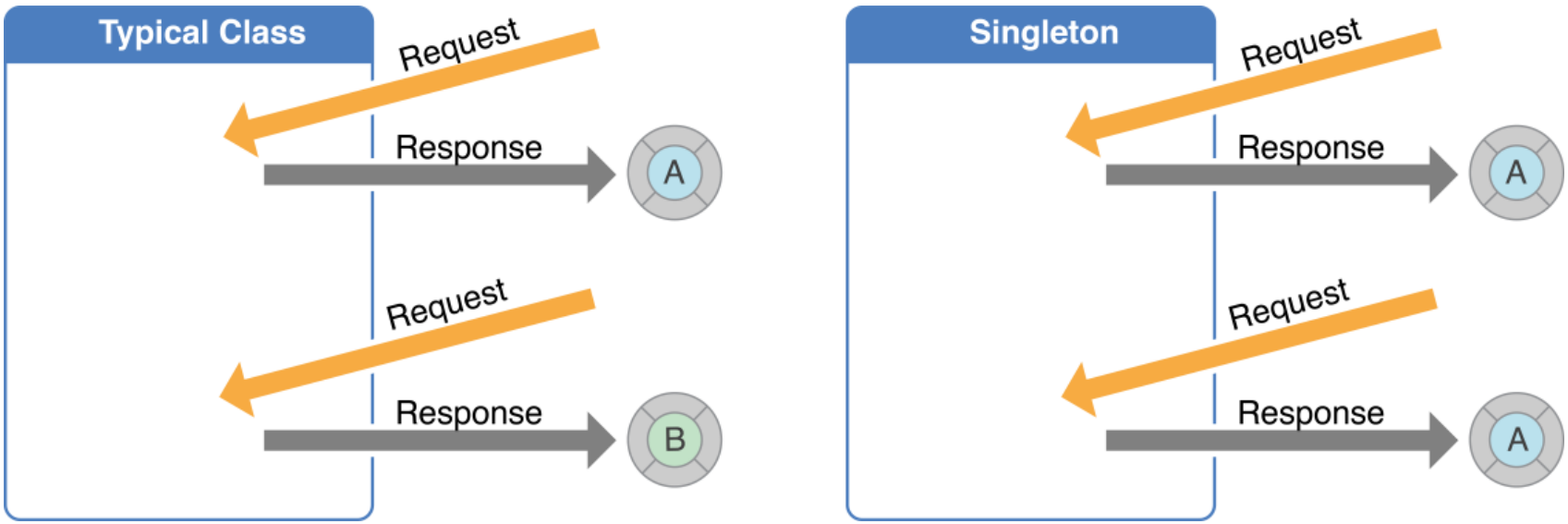
Opis

Singleton zapewnia występowanie maksymalnie jednej instancji danej klasy (tworzenie obiektu następuje przy pierwszej próbie użycia) oraz umożliwia globalny dostęp do stworzonego obiektu. Zazwyczaj wykorzystywany jest do przechowywania konfiguracji aplikacji oraz utrzymywania połączenia z bazą danych. Przy implementacji należy pamiętać o środowisku wielowątkowym i trudnościach z testowaniem.

Singleton - schemat działania

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

https://pl.wikipedia.org/wiki/Singleton



https://developer.apple.com/

Singleton w Javie

```
public class Conf {
    private static final Conf INS = new
        Conf();

    private Conf() {
    }

    public static Conf getInstance() {
        return INS;
    }
}
```

Singleton w Javie z lazy loading

```
public class Conf {
    private static Conf ins = null;

    private Conf() {
    }

    public static Conf getInstance() {
       if(ins == null) {
            ins = new Conf();
        }
        return ins;
    }
}
```

Singleton w Javie z lazy loading 2

```
public class Conf {
    private static Conf ins = null;

    private Conf() {
    }

    public static Conf getInstance() {
      if(ins == null) {
          synchronized(Conf.class) {
              ins = new Conf();
          }
       }
       return ins;
    }
}
```

Singleton w Javie z lazy loading 3

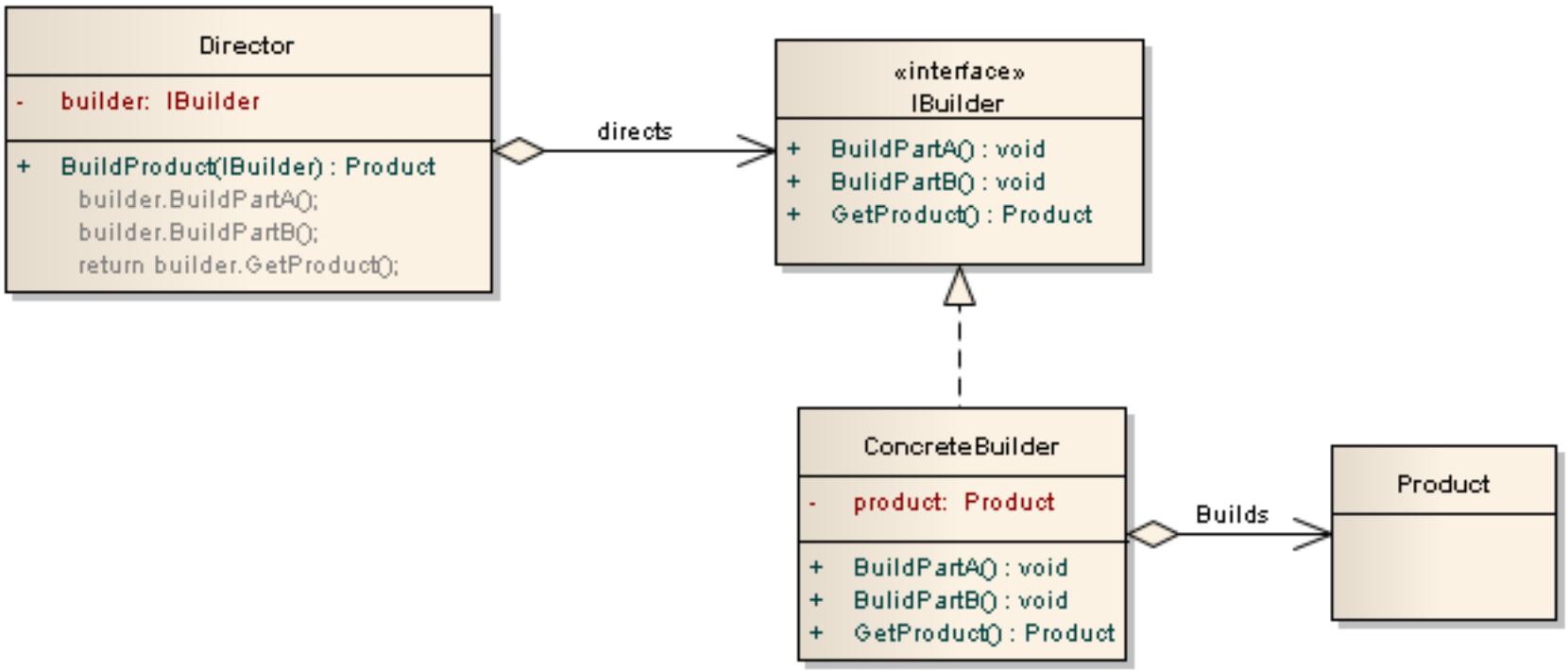
```
public enum Conf {
   INSTANCE;
}
```

Budowniczy

Opis

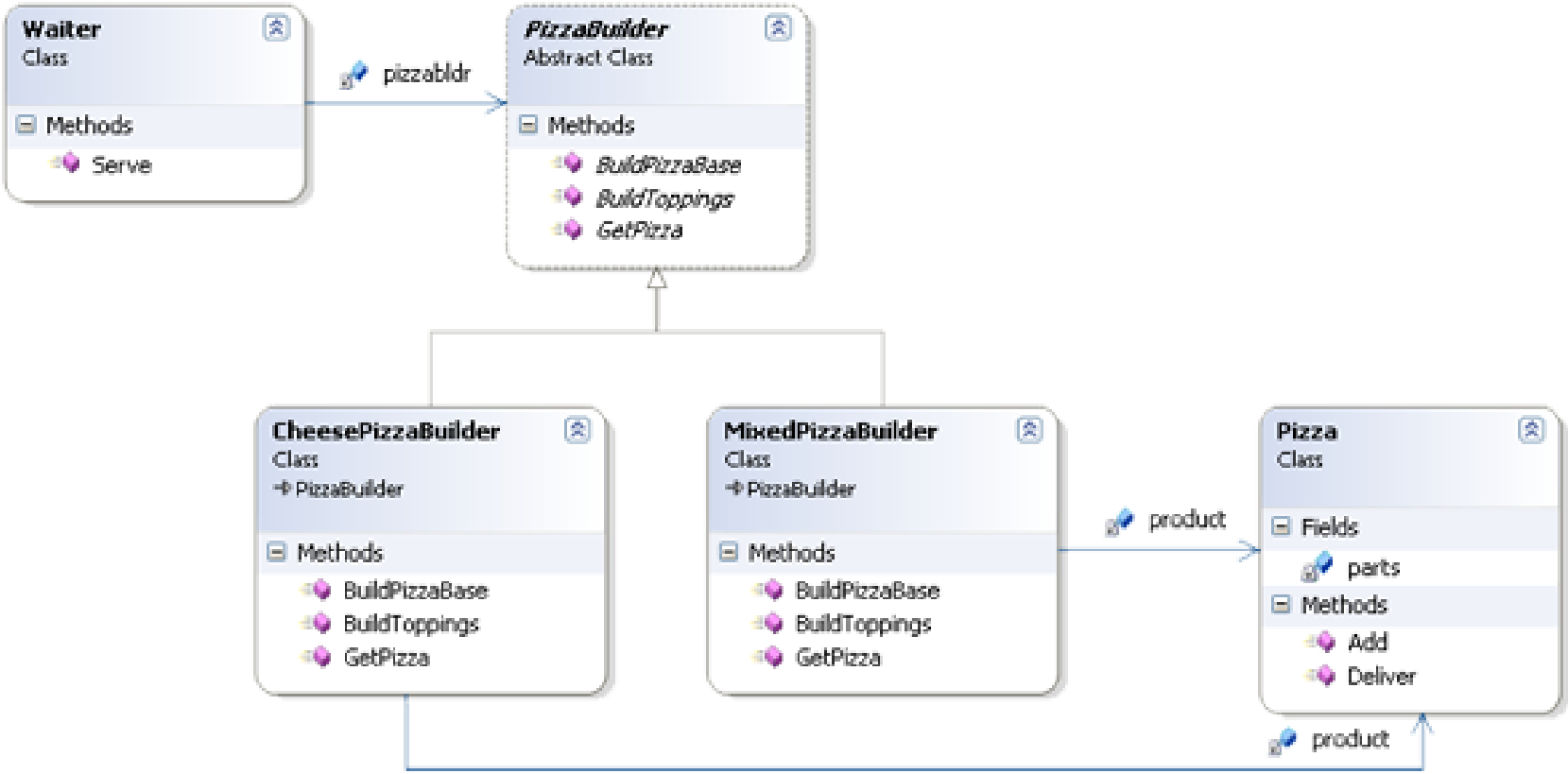
Budowniczy (ang. builder) zakłada rozdzielenie sposobu tworzenia obiektów od ich reprezentacji - proces tworzenia jest dekomponowany na kilka pomniejszych etapów, które są implementowane na różne rodzaje (algorytm konstrukcji obiektu musi być niezależny od składowych obiektu i sposobu ich łączenia). Reasumując możliwe jest np. stworzenie różnych produktów bez zmiany kodu metody tworzącej i struktury produktu.

Budowniczy - UML



http://devlake.com

Budowniczy - przykład



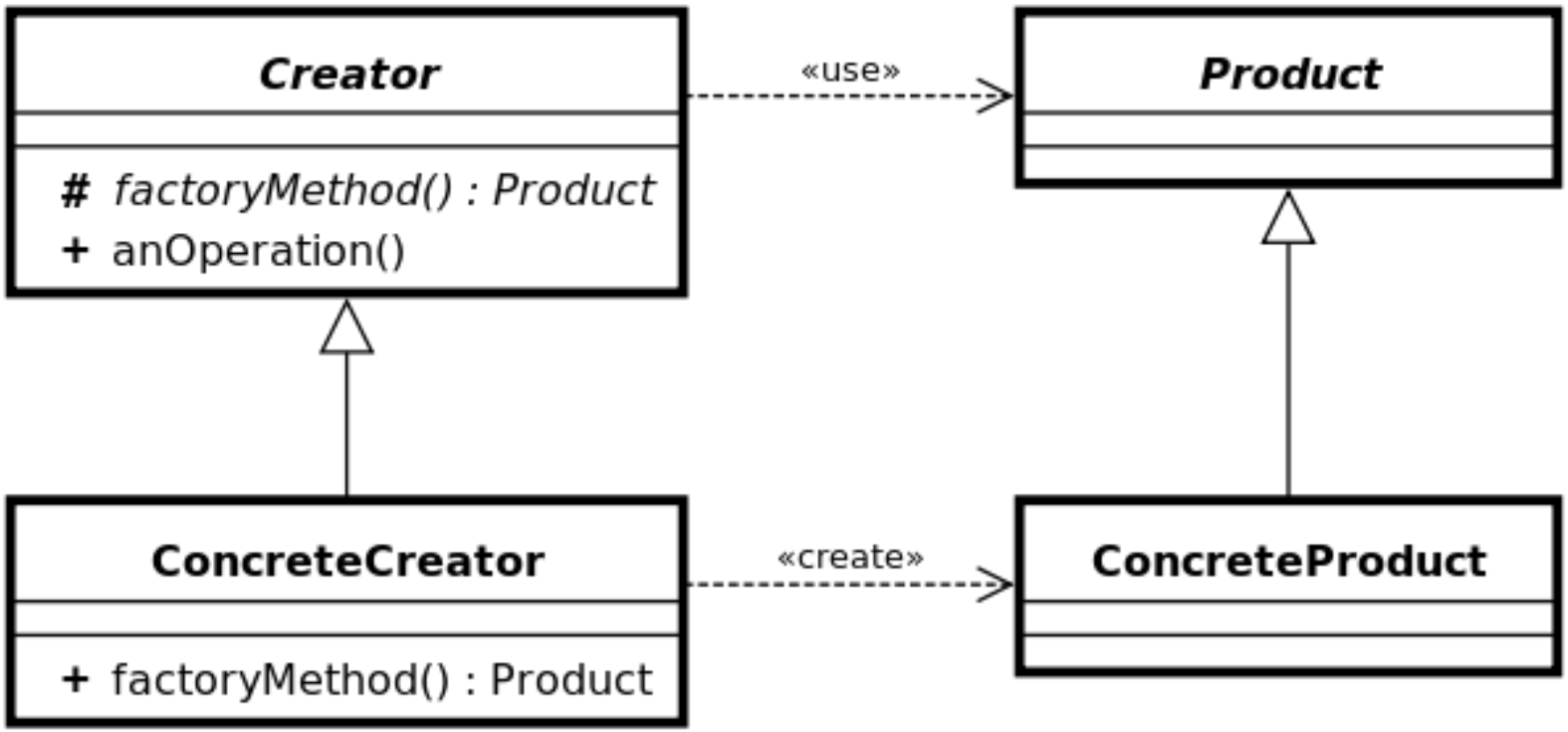
http://go4expert.com

Metoda wytwórcza

Opis

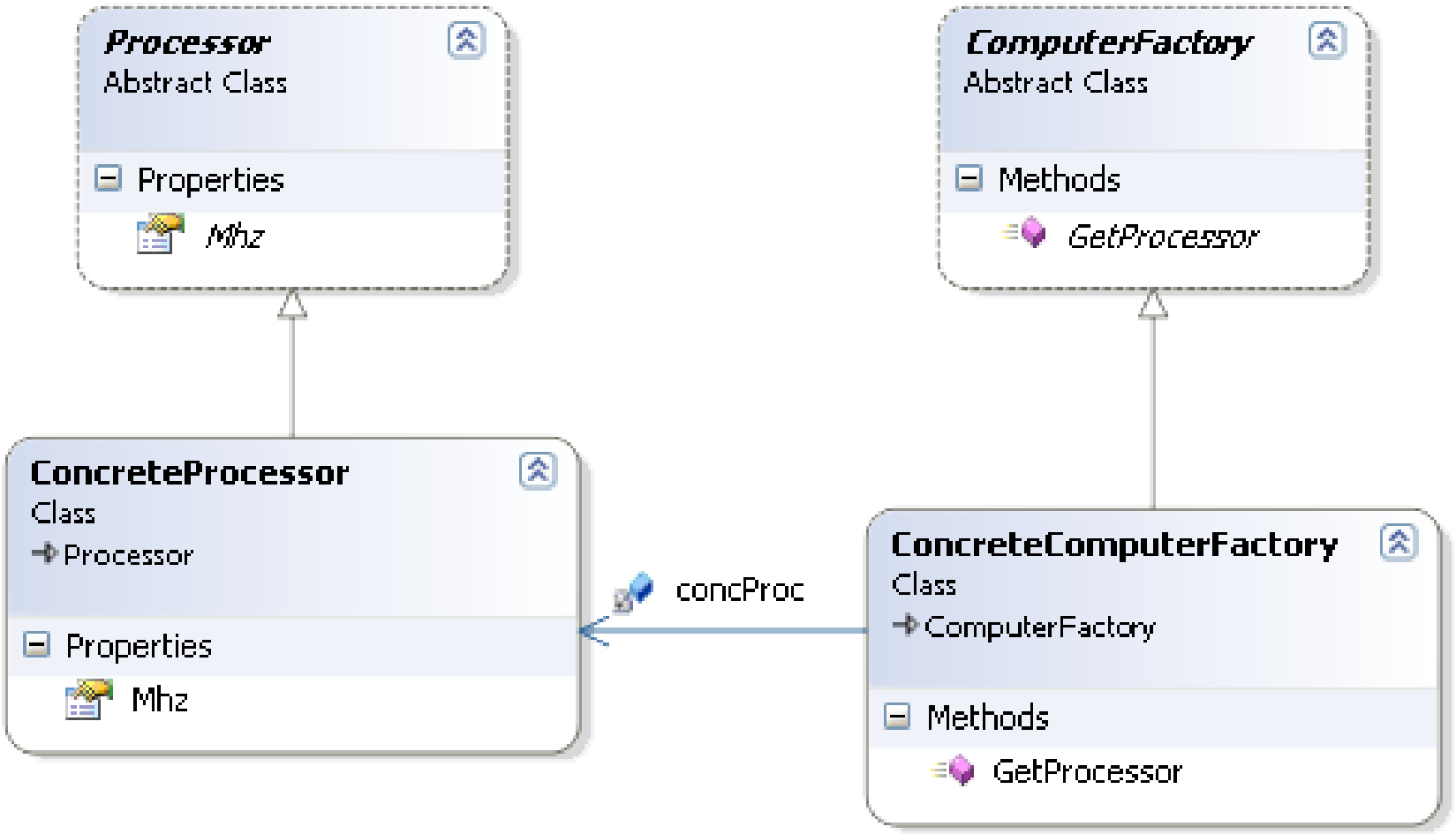
Metoda wytwórcza (ang. factory method) umożliwia tworzenie nowych obiektów, które są nieokreślone, ale związane wspólnym interfejsem. Zapewnia hermetyzację procesu tworzenia obiektów (jedynie metoda wytwórcza zna szczegóły implementacyjne niezbędne do stworzenia obiektu) pozwalając na rozszerzenie funkcjonalności w przyszłości. W przeciwieństwie do budowniczego nie zmieniamy zawartości złożonego produktu, a jedynie wybieramy między różnymi klasami produktu.

Metoda wytwórcza - UML



http://upload.wikimedia.org

Metoda wytwórcza - przykład



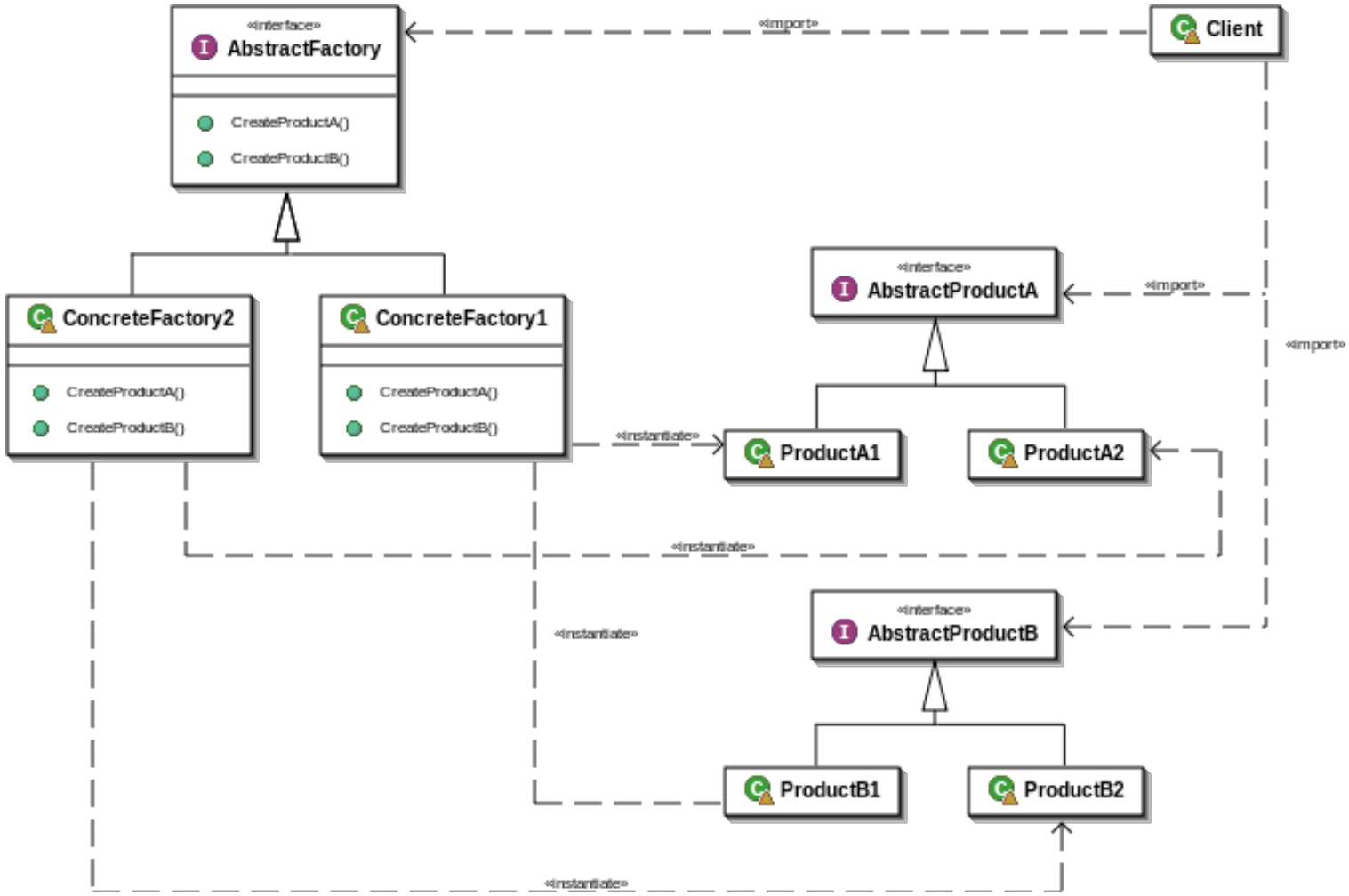
http://go4expert.com

Fabryka abstrakcyjna

Opis

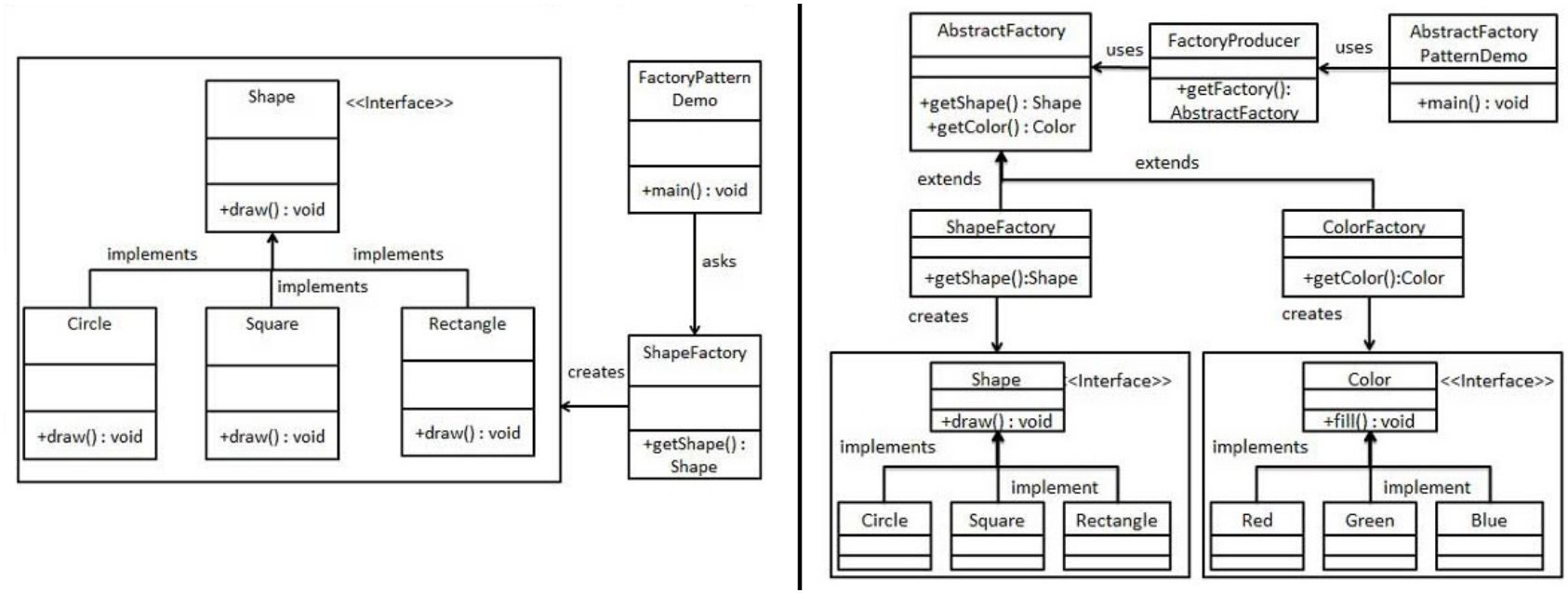
Fabryka abstrakcyjna (ang. abstract factory) udostępnia interfejs do tworzenia różnych obiektów danego typu, bez specyfikowania ich klas. W przeciwieństwie do metody wytwórczej (tworzącej jeden produkt), fabryka abstrakcyjna tworzy rodzinę powiązanych produktów.

Fabryka abstrakcyjna - UML



https://upload.wikimedia.org

Metoda wytwórcza, a fabryka abstrakcyjna



http://stackoverflow.com

Zadanie

1. Zaimplementuj wybrany wzorzec kreacyjny.

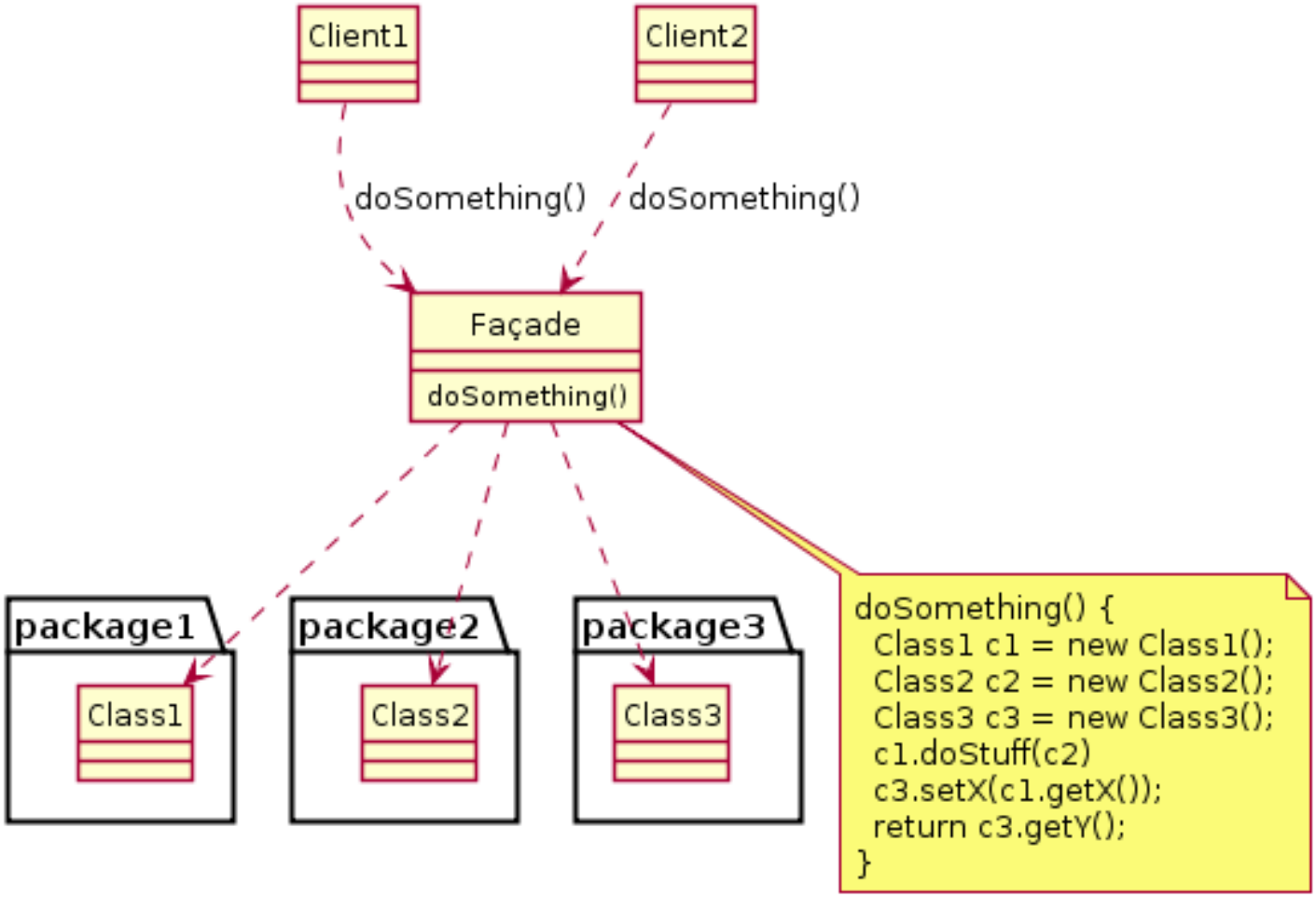
Wzorce strukturalne

Fasada

Opis

Fasada (ang. facade) wykorzystywany jest w celu ujednolicenia i uproszczenia dostępu do złożonego systemu tworząc interfejs wyższego poziomu.

Fasada - UML i przykład



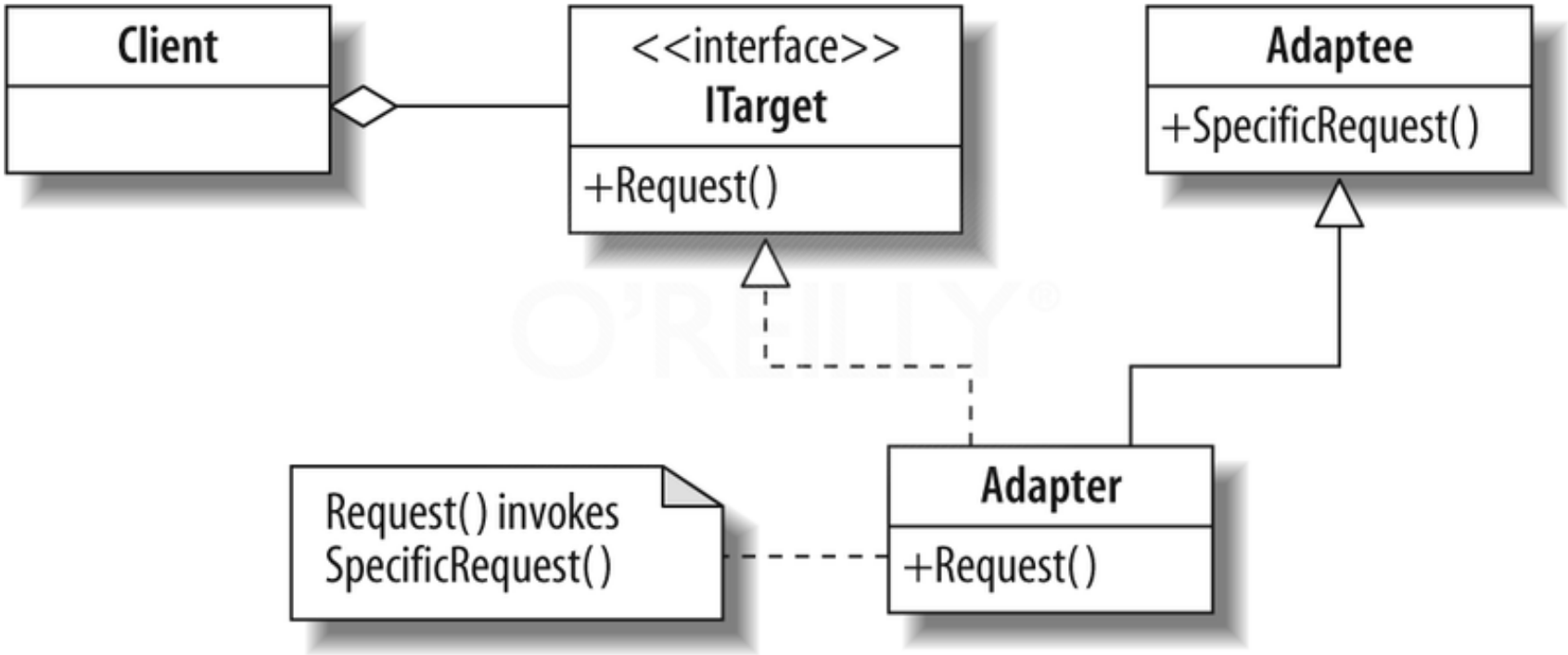
https://en.wikipedia.org/wiki/Facade_pattern

Adapter

Opis

Adapter (ang. wrapper) umożliwia współpracę dwóm klasom, które mają niekompatybilne interfejsy, poprzez przekształcenie interfejsu jednej z klas na interfejs drugiej.

Adapter - przykład



http://i-msdn.sec.s-msft.com

Zadanie

1. Zaimplementuj wybrany wzorzec strukturalny.

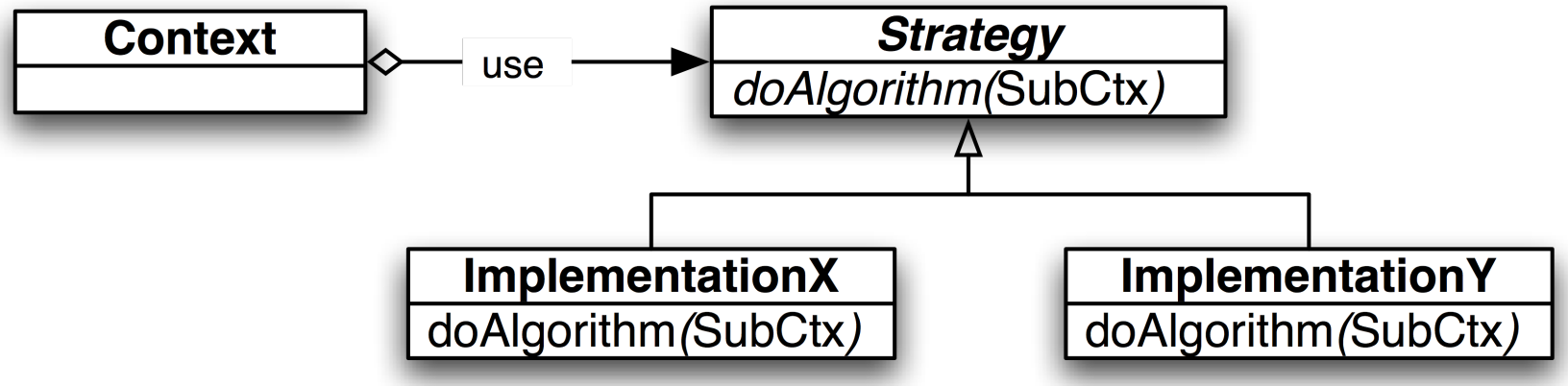
Wzorce czynnościowe

Strategia

Opis

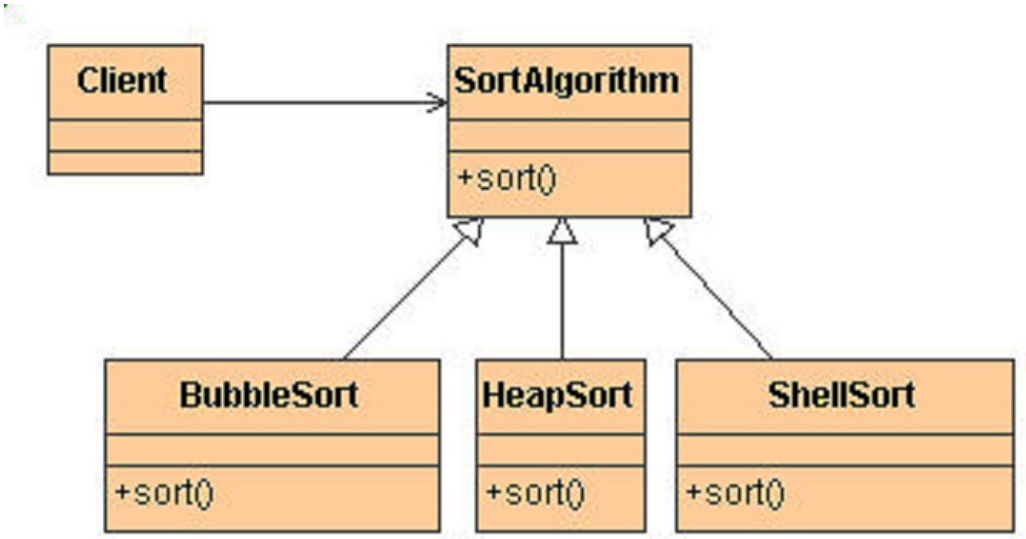
Strategia (ang. strategy) umożliwia wymienne stosowanie algorytmów w trakcie działania programu. Algorytmy zostają opakowane w oddzielne klasy.

Strategia - UML



http://anirudhbhatnagar.files.wordpress.com

Strategia - przykład



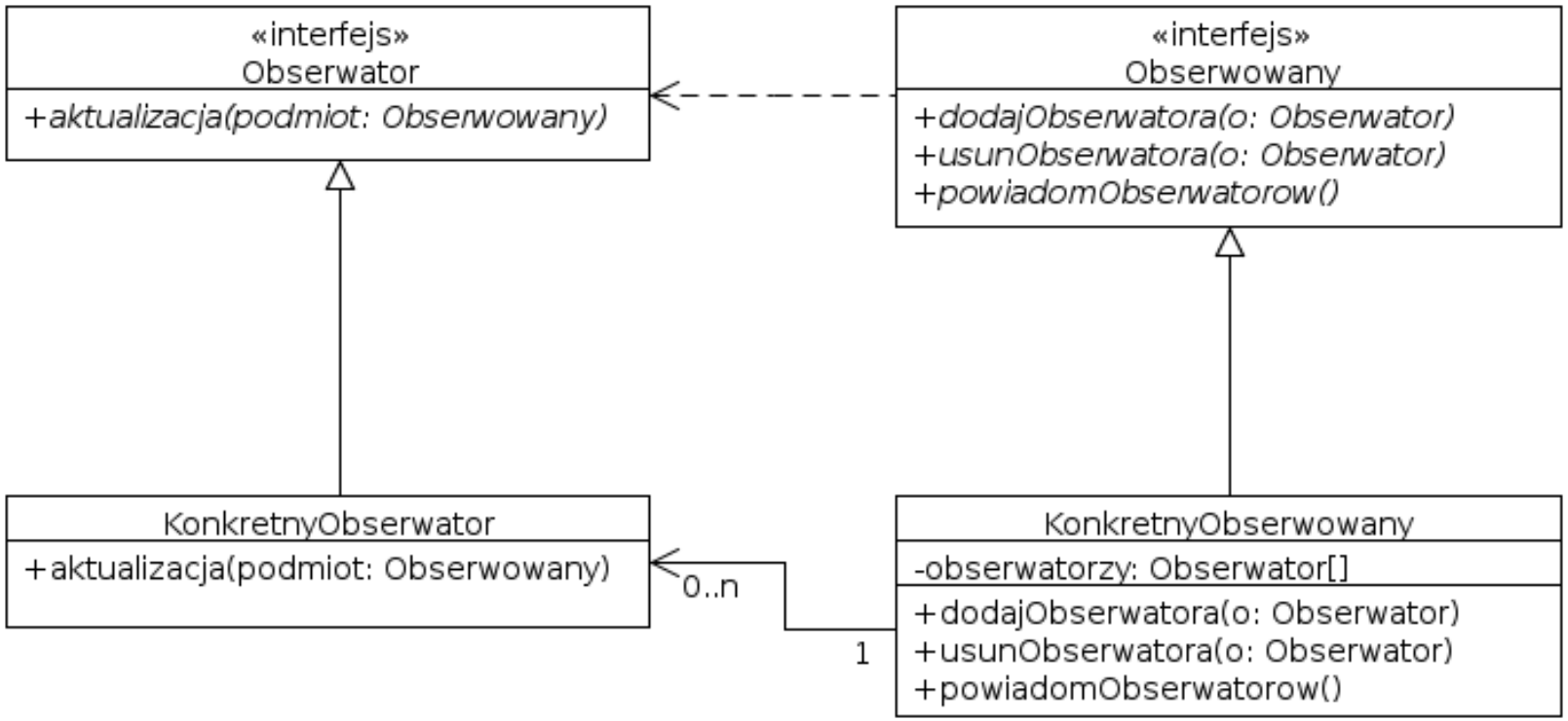
http://1.bp.blogspot.com

Obserwator

Opis

Obserwator (ang. observer) opiera się na powiadamianiu zainteresowanych obiektów (obserwatorów) o zmianie statusu obserwowanego obiektu.

Obserwator - UML



http://pl.wikipedia.org

Zadanie

1. Zaimplementuj wybrany wzorzec czynnościowy.

Wzorce architektoniczne

Opis

Wzorce architektoniczne (ang. Architectural Patterns) określają ogólną strukturę systemu informatycznego oraz jego elementy, wraz z zakresem funkcji przez nie realizowanych i zasad komunikacji między nimi. W ramach przedmiotu zajmiemy się wzorcami MVC, MVP oraz MVVM, które są traktowane - w niektórych źródłach - jako wzorce architektoniczne lub wzorce projektowe warstwy prezentacji.

Opis

Model-Widok-Kontroler (ang. Model-View-Controller) zakłada podział aplikacji na trzy warstwy:

- Model - zawiera logikę aplikacji.
- Widok - wyświetla interfejs użytkownika aplikacji. Posiada referencje do modeli, z których pobiera dane w momencie otrzymania od kontrolera żądania odświeżenia.
- Kontroler - obsługuje zdarzenia. Może zmienić stan modelu, odświeżyc widok i przełączyć sterowanie na inny kontroler.

MVP

Opis

Model-Widok-Prezenter (ang. Model-View-Presenter) jest rozwinięciem MVC i zakłada podział aplikacji na trzy warstwy:

- Model - zawiera logikę aplikacji.
- Widok - wyświetla interfejs użytkownika aplikacji.
- Prezenter - wysyła zapytanie do modelu i przetwarza otrzymane dane, po czym przekazuje je do widoku.

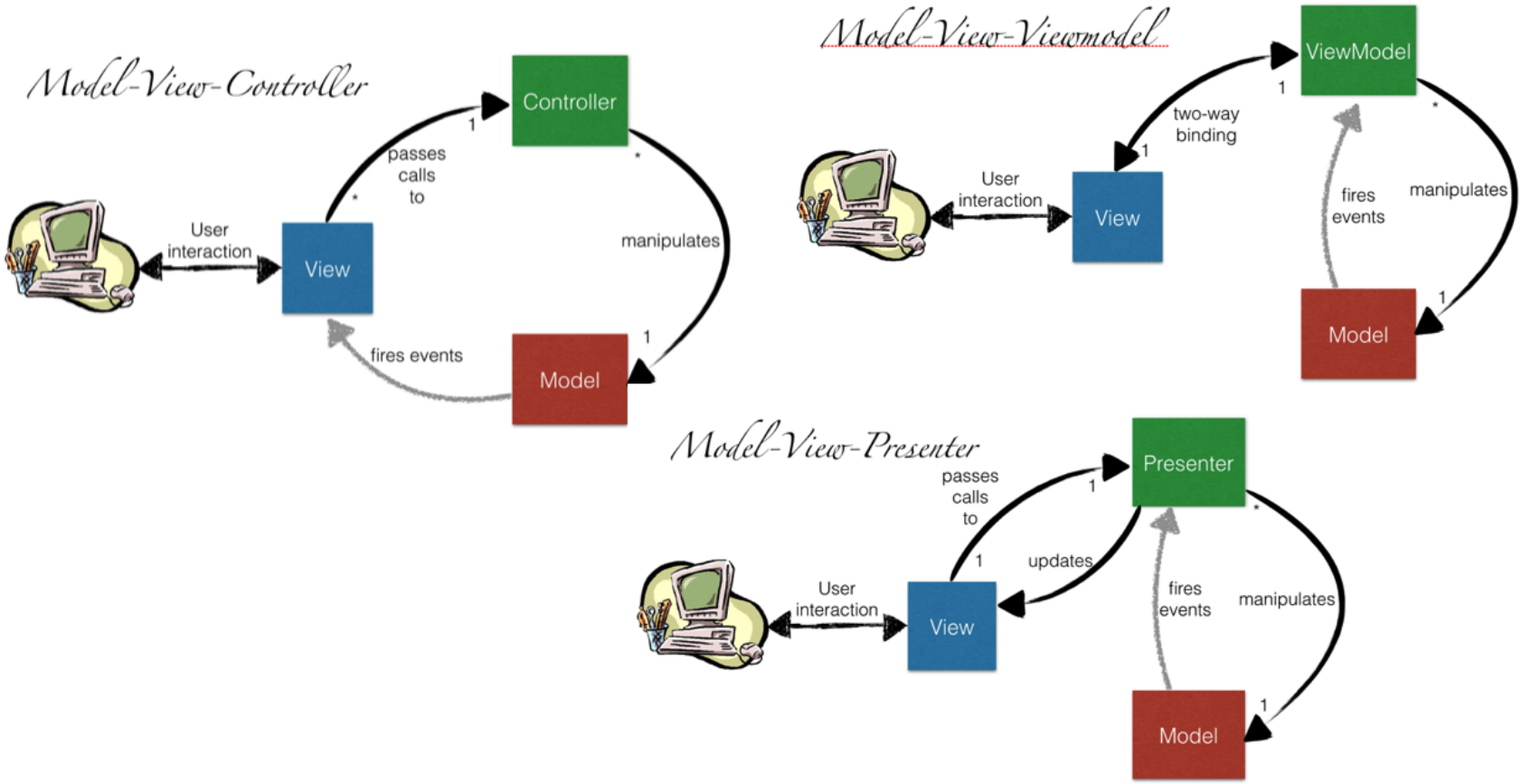
MVVM

Opis

Model-Widok-Widok Modelu (ang. Model-View-ViewModel) jest rozwinięciem MVC i zakłada podział aplikacji na trzy warstwy:

- Model - zawiera logikę aplikacji.
- Widok - wyświetla interfejs użytkownika aplikacji.
- Widok Modelu - reaguje na reakcje użytkownika i odpowiada za pobieranie oraz modyfikowanie informacji w modelu, a także za aktualizowanie widoku.

Porównanie wzorców MV*



http://beyondjava.net

Dziękuję za uwagę