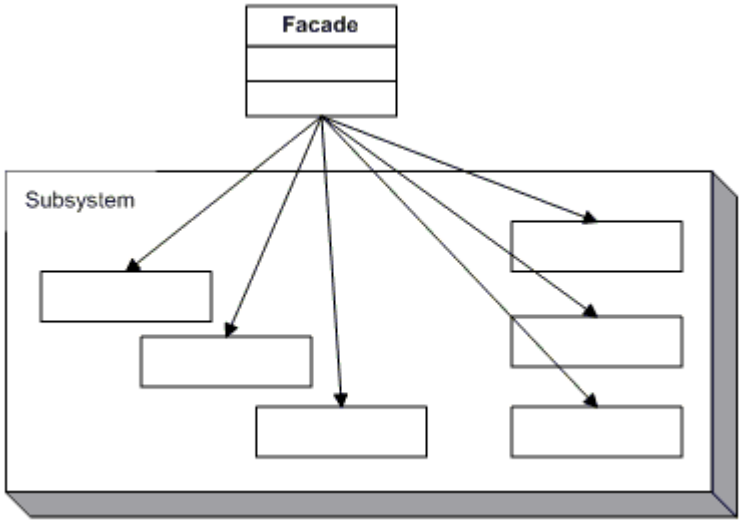
Projektowanie obiektowe oprogramowania
Wykład 5 – wzorce strukturalne
Wiktor Zychla 2021

1 Facade

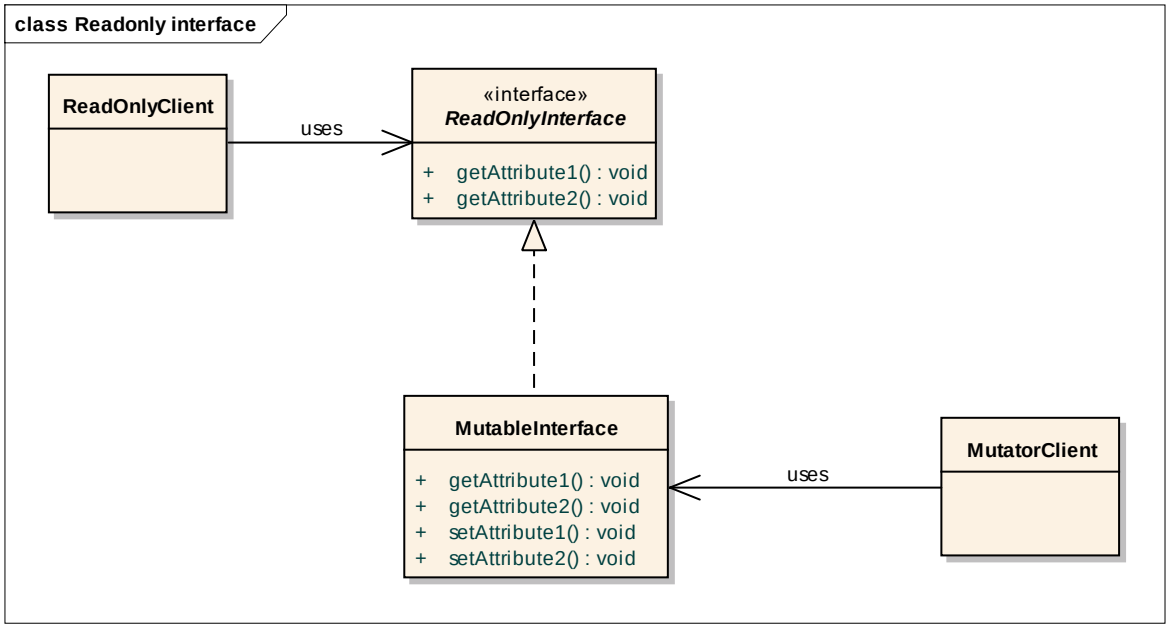
Motto: uproszczony interfejs dla podsystemu z wieloma interfejsami




```
class SmtpFacade {
      public void Send( string From, string To,
                        string Subject, string Body,
                        Stream Attachment, string AttachmentMimeType );
}
```

2 Read-only interface

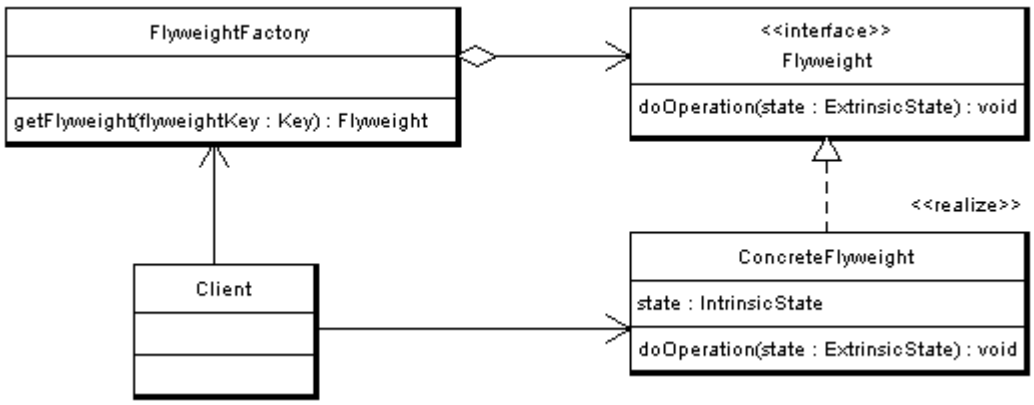
Motto: interfejs do odczytu dla wszystkich, a do zapisu tylko dla wybranych



Przykład: ReadOnlyCollection, AsReadOnly()

3 Flyweight

Motto: Efektywne zarządzanie wieloma drobnymi obiektami
Kojarzyć: Object Pool + immutable + bardzo dużo danych – zapamiętać przykład z wykładu Board vs Checker



Flyweight w implementacji przypomina Object Pool. Różnica jest taka, że Pool utrzymuje pulę **rozłącznych** obiektów, zwraca za każdym razem inną instancję, a po użyciu instancja ta wraca do puli. Motywacją dla Object Pool jest bardzo kosztowne tworzenie instancji.

W przypadku Flyweight jest inaczej – motywacją dla tego wzorca jest bowiem chęć oszczędzania pamięci w sytuacji, gdy do utworzenia jest naprawdę duża liczba potencjalnych obiektów, **zbyt** duża żeby reprezentować je w sposób klasyczny. FlyweightFactory utrzymuje więc dużo mniejszą pulę obiektów – taką, w której wyznacznikiem tożsamości obiektu jest stan, który można współdzielić między „rzekomo" różnymi instancjami. Klient dostaje wiele obiektów, a w rzeczywistości za każdym razem kiedy prosi o obiekt o tym samym „kluczu", dostaje tę samą instancję.

Takie podejście rodzi oczywiście problem identyfikacji takiej części stanu obiektu która może być współdzielona (*intrinsic*) i takiej która nie może być współdzielona (*extrinsic*) i jej przechowaniem w optymalny dla pamięci sposób zajmuje się albo FlyweightFactory albo jeszcze inny obiekt.

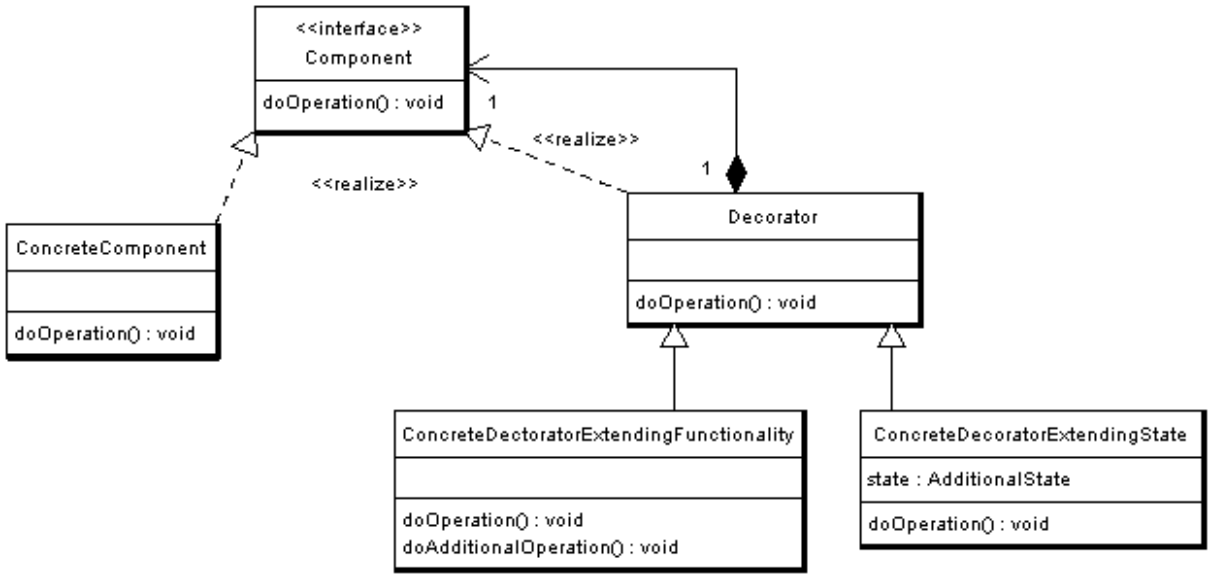
Na przykład (przykład z wykładu) wyobraźmy sobie planszę do gry w warcaby o wymiarach 1mln na 1mln pól. Gdyby taka plansza była zaimplementowana tak, żeby utrzymywać instancje obiektów bierek dla każdego pola planszy, zużycie pamięci byłoby ogromne.

Zamiast tego, w klasie planszy reprezentacja stanu planszy jest jakaś optymalna (na przykład za pomocą tablicy bajtów). Ale klient nadal potrzebuje obiektu – bierki, chociażby po to żeby wywołać na nim metody do rysowania.

Tu wkracza Flyweight. Okazuje się, że z punktu widzenia silnika graficznego, bierki są tak naprawdę dwie różne – biała i czarna. FlyweightFactory (może być osobną klasą, a może to być sama klasa planszy) zapamiętuje więc tylko **dwie** różne instancje klasy bierki, każda z nich ma zapamiętaną tę współdzieloną (*intrinsic*) część stanu (tu: kolor). Reszta stanu (*extrinsic*), czyli np. pozycja na planszy, przychodzi z zewnątrz, tu: z klasy planszy. Metoda do rysowania bierki będzie więc miała parametr do przekazania tego zewnętrznego, niewspółdzielonego stanu.

4 Decorator

Motto: Dynamicznie rozszerzanie odpowiedzialności obiektów (alternatywa dla podklas)
Kojarzyć: w bibliotekach standardowych technologii przemysłowych od lat używa się wzorca Decorator do implementacji **podsystemu strumieni**



5 Proxy

Motto: substytut (zamiennik) obiektu w celu sterowania dostępem do niego

- Proxy zdalne – reprezentant lokalny obiektu zdalnego

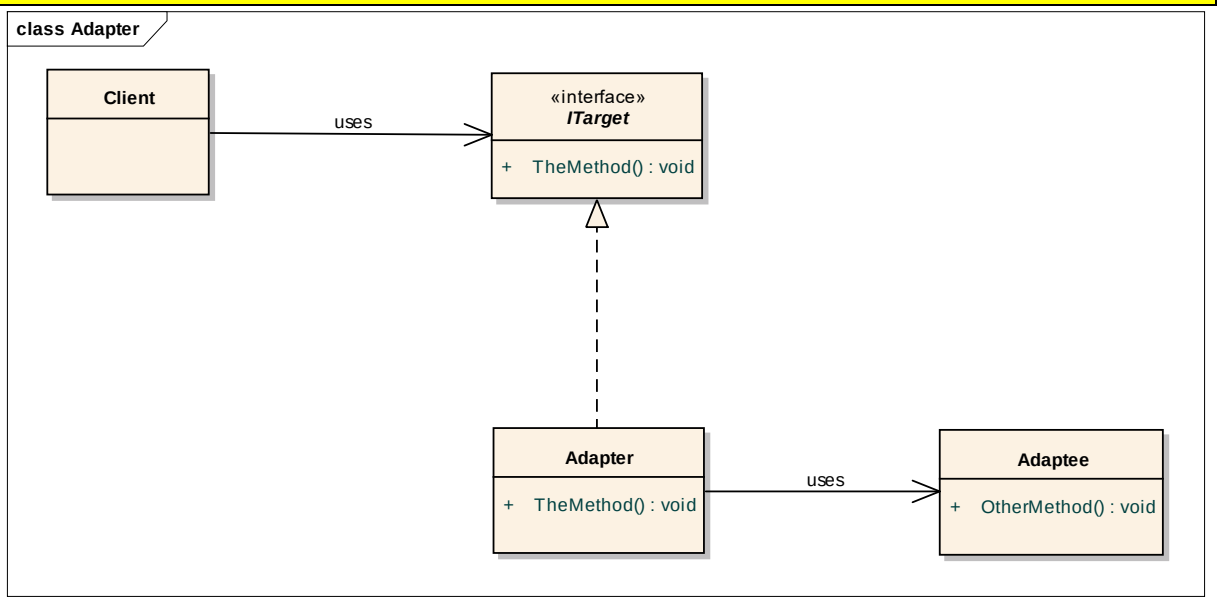
- Proxy wirtualne – tworzy kosztowny obiekt na żądanie
- Proxy ochraniające – kontroluje dostęp do obiektu
- Proxy logujące – loguje dostęp do obiektu

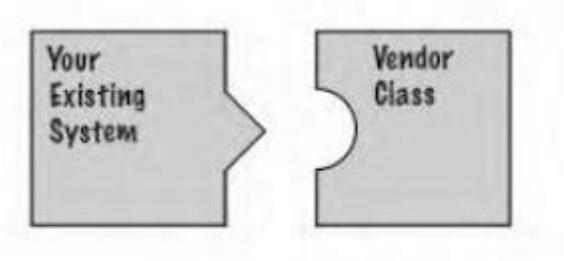
Przykład virtual proxy z biblioteki standardowej .NET - Lazy<T>.

Uwaga! Struktura Proxy i Decoratora może wydawać się podobna. Różnice są następujące:
- Każdy z Decoratorów może dodawać nowe, specyficzne dla siebie operacje/informacje; Proxy nigdy nie zmienia (nie rozszerza) interfejsu obiektu
- Proxy z zasady nie jest przeznaczone do stosowania „rekursywnego”; Dekorator przeciwnie

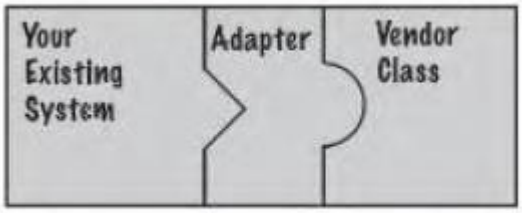
6 Adapter

Motto: uzgadnianie niezgodnych interfejsów





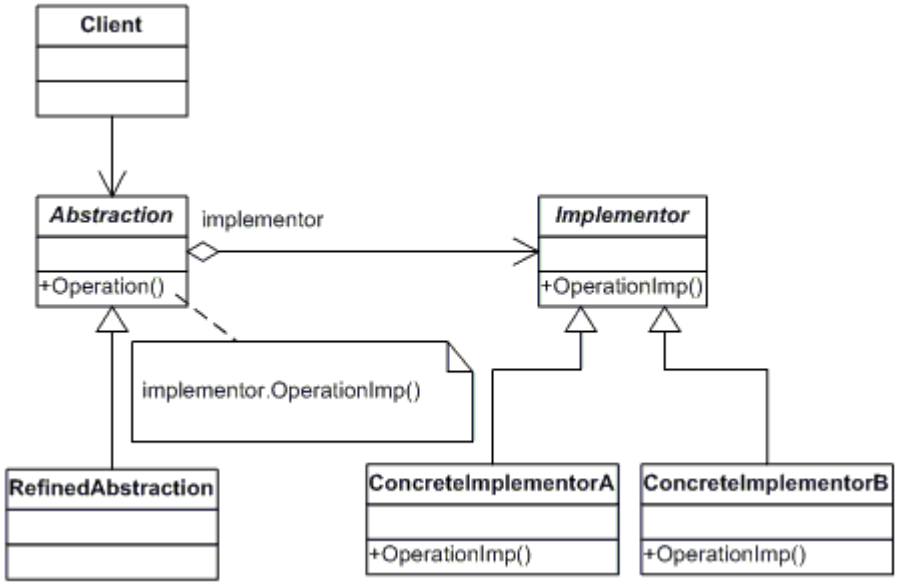
Not Compatible? We are in trouble



God Adapter Save our Life

7 Bridge

Motto: SRP + ISP + DIP dla hierarchii obiektowej o dwóch stopniach swobody; oddzielenie abstrakcji od implementacji



8 Literatura

[1] Gamma, Helm, Johnson, Vlissides – Wzorce projektowe
[2] Martin, Martin – Zasady, wzorce I praktyki zwinnego wytwarzania oprogramowania w C#
[3] Grand, Merrill – Wzorce projektowe
[4] Freeman, Freeman, Sierra, Bates – Head First Design Patterns
[5] http://www.oodesign.com