Wzorce Projektowe

Jakub Jaśków

15 października 2025

Plan prezentacji

1. Co to są wzorce projektowe i po co są stosowane?
2. Podział na rodzaje
3. Najczęściej spotykane wzorce z przykładami
4. Bibliografia

Definicja wzorca projektowego

- **Wzorzec projektowy** (ang. *design pattern*) – uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych.
- Termin "wzorzec projektowy"został wprowadzony przez Kenta Becka oraz Warda Cunninghama w 1987, spopularyzowany w 1995 roku w książce *Wzorce projektowe: Elementy oprogramowania obiektowego wielokrotnego użytku* autorstwa *Ericha Gamma*, *Richarda Helma*, *Ralpha Johnsona* oraz *Johna Vlissidesa*, zwanych też "Czwórką"lub "Gangiem Czworga"(ang. *Gang of Four*).

Zastosowanie wzorca projektowego

- **Cel**: ułatwienie opisu rozwiązania potencjalnego problemu, doprecyzowanie myśli
- **Przykład rozmowy**: "Do poradzenia sobie z problemem zmieniających się interfejsów dla zewnętrznych usług zastosujemy Adaptery generowane przez Singletonową Fabrykę"

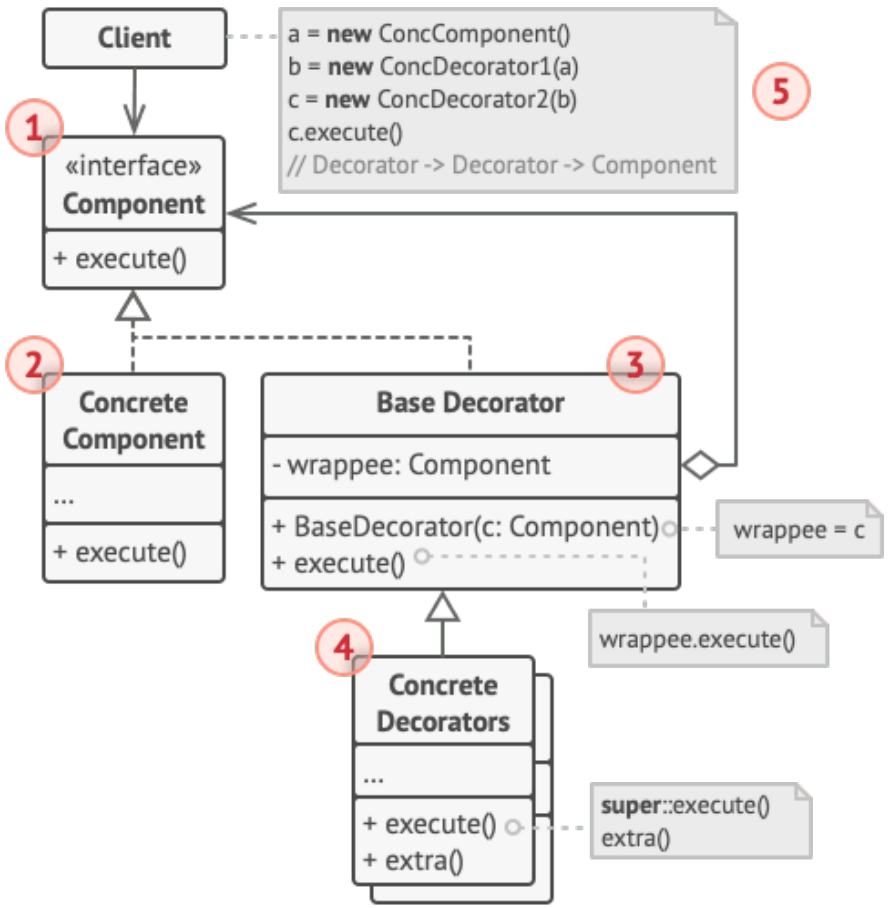
Podział wzorców

- **Kreacyjne** - pozyskiwanie obiektów zamiast bezpośredniego tworzenia instancji klas.
- **Strukturalne** - składanie obiektów i klas w większe struktury, przy zachowaniu elastyczności i efektywności.
- **Behawioralne** - charakteryzują odpowiedzialności oraz oddziaływania pomiędzy poszczególnymi klasami i obiektami.

Podział wzorców według "Czwórki"

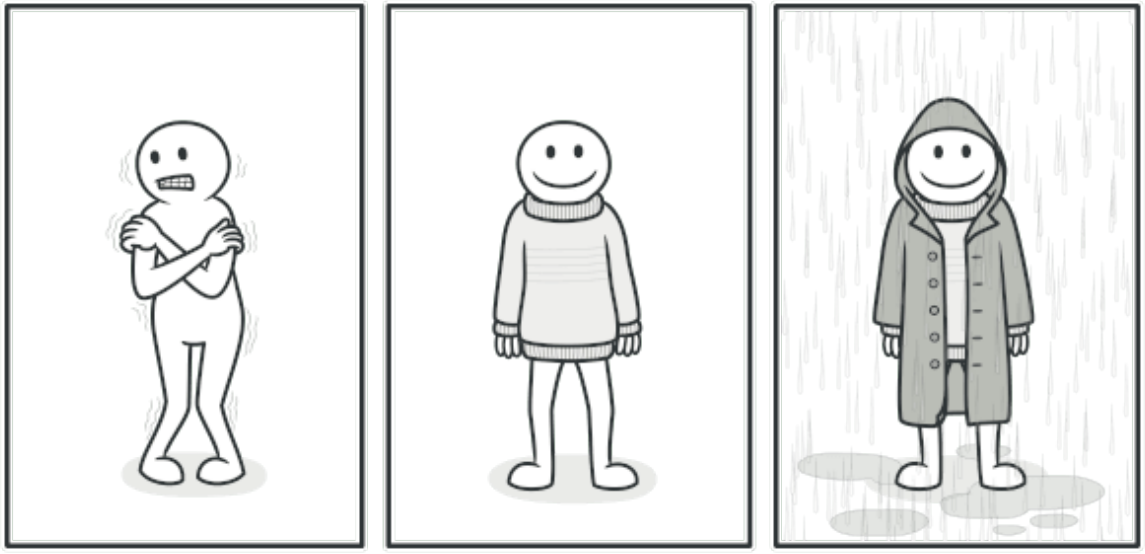
		Przeznaczenie		
		Kreacyjne	**Strukturalne**	**Behawioralne**
Zakres	**Klasa**	FactoryMethod	Adapter	Interpreter Template Method
	Obiekt	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Proxy Flyweight	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Dekorator (*ang. decorator*)

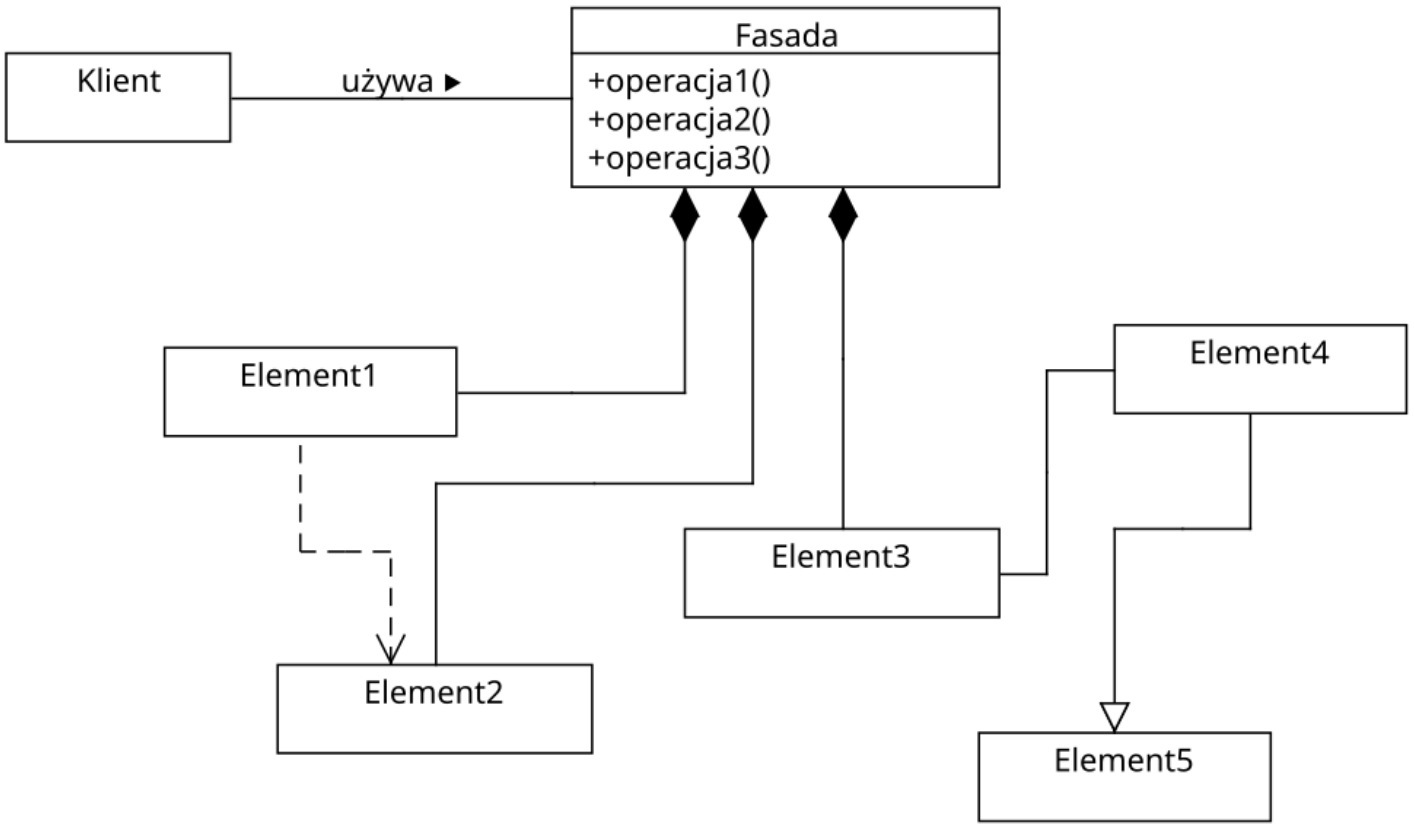


Dekorator

- Opakowanie oryginalnej klasy w nową klasę "dekorującą"
- Oryginalny obiekt przekazany jako parametr konstruktora dekoratora
- Metody dekoratora wywołują metody oryginalnego obiektu i dodatkowo implementują nową funkcjonalność
- Alternatywa dla dziedziczenia. Pozwala na rozszerzenie zachowania klasy w trakcie działania programu

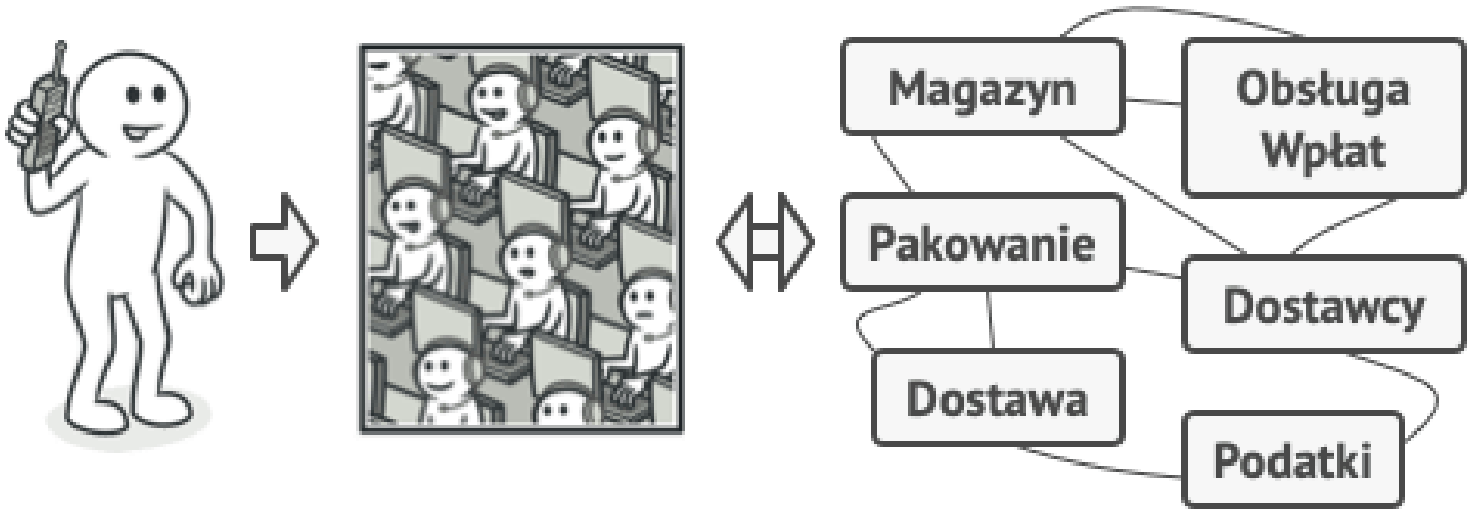


Fasada (*ang. Facade*)



Fasada

- **Problem**: Potrzebny jest wspólny, jednorodny interfejs do zbioru implementacji lub interfejsów (np. w ramach podsystemu). Chcemy ograniczyć niepożądane sprzężenie ze składnikami podsystemu oraz zabezpieczyć się przed jego zmianami.
- **Rozwiązanie**: Zdefiniuj pojedynczy punkt dostępu dla całego systemu – obiekt fasadę, który opakowuje podsystem.

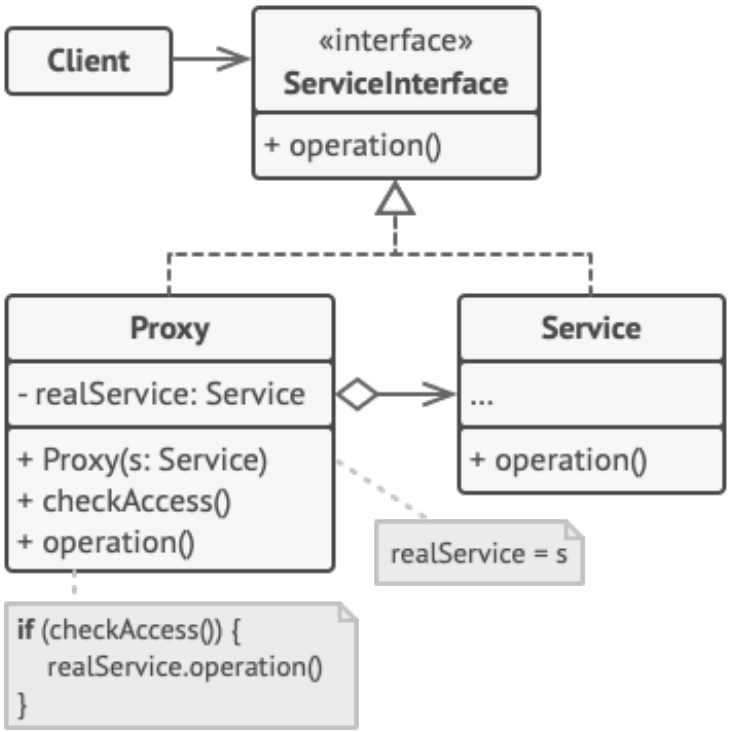


Fasada - zalety

- Ukrywanie złożoności tworzonego przez siebie systemu przez dostarczenie udokumentowanego, publicznego API. Korzyścią z zastosowania w tym przypadku fasady jest to, że programiści korzystający z systemu muszą przyswoić sobie tylko API fasady a nie wszystkich obiektów systemu.
- W ogólności wzorzec fasady pozwala wykorzystać podzbiór możliwości skomplikowanego systemu w prostszy, specyficzny dla danego zastosowania sposób.

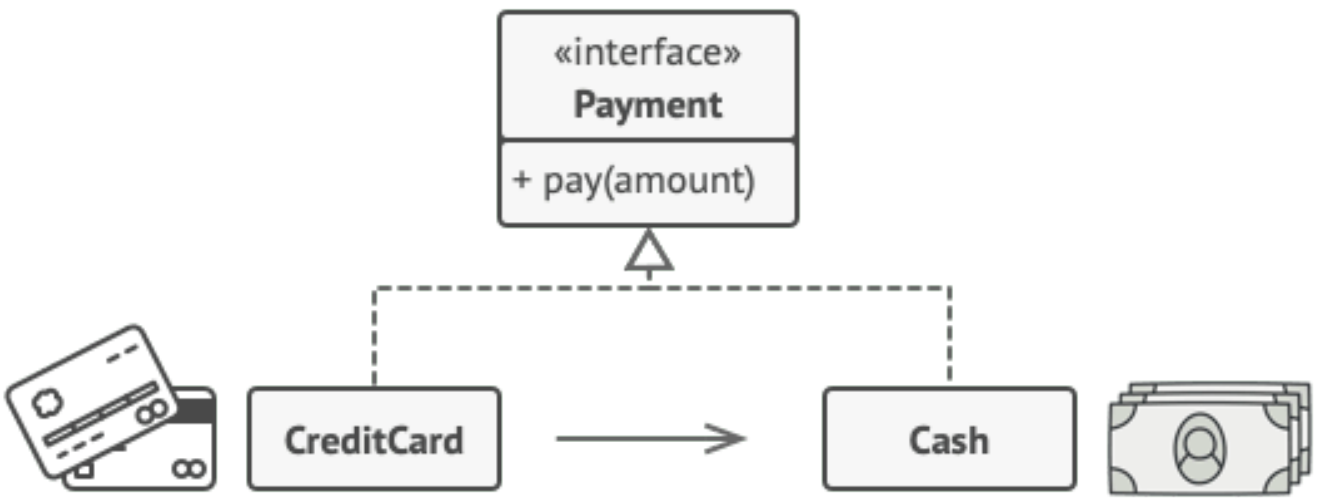
Pełnomocnik (*ang. Proxy*)

- Pełnomocnik nadzoruje dostęp do pierwotnego obiektu, pozwalając na wykonanie jakiejś czynności przed lub po przekazaniu do niego żądania.



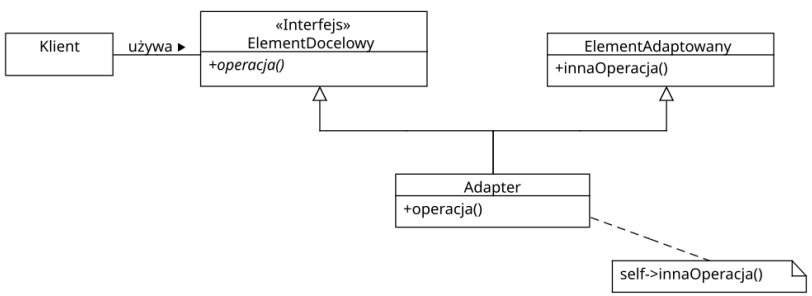
Pełnomocnik - zalety

- Leniwa inicjalizacja
- Kontrola dostępu
- Lokalne uruchamianie zdalnej usługi
- Prowadzenie dziennika żądań
- Możliwość likwidacji niepotrzebnego obiektu

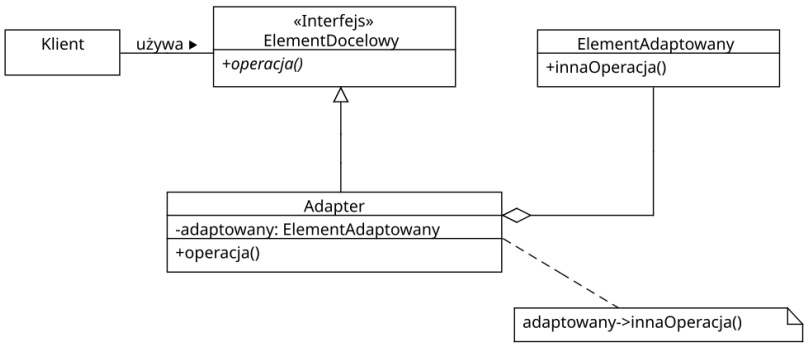


Adapter

- **Adapter** jest strukturalnym wzorcem projektowym pozwalającym na współdziałanie ze sobą obiektów o niekompatybilnych interfejsach.
- **Przykład**: działamy na plikach XML, natomiast potrzebna nam biblioteka działa tylko na plikach JSON



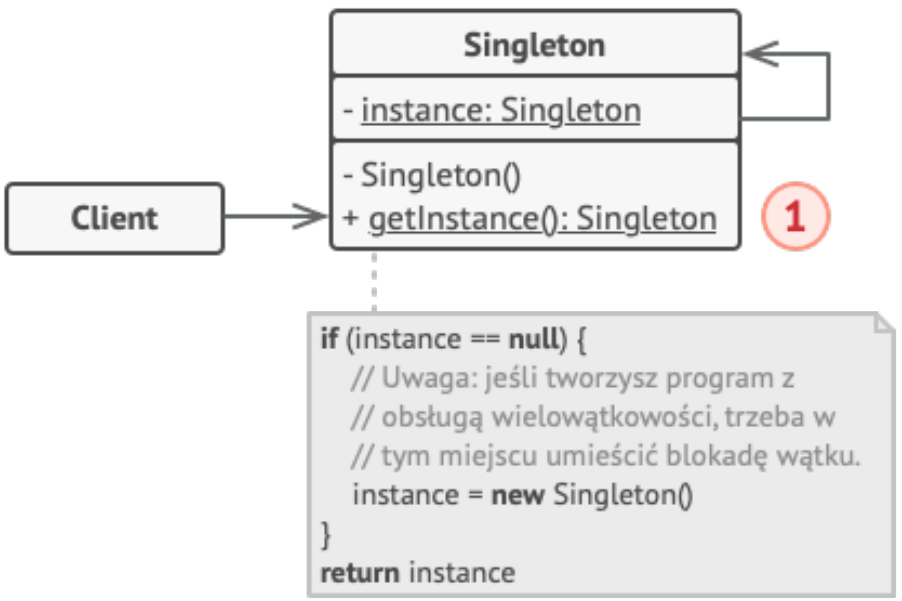
Rysunek: Adapter klasowy

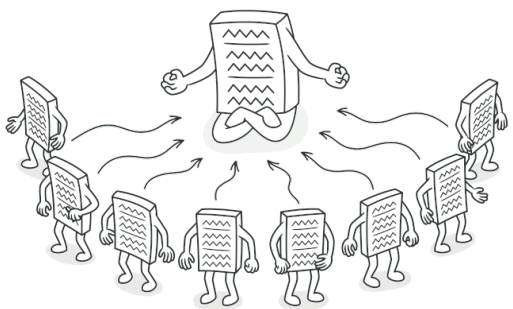


Rysunek: Adapter obiektowy

Singleton

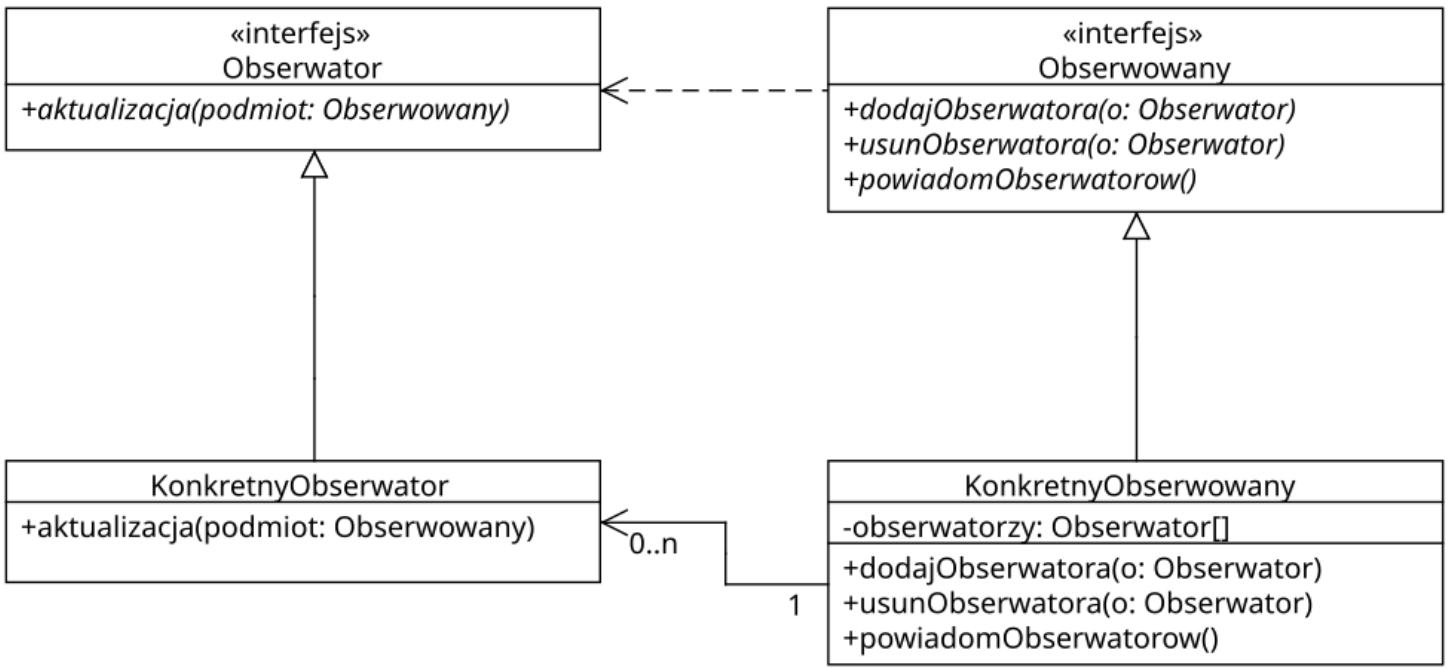
- Zapewnia istnienie wyłącznie jednej instancji danej klasy.
- Pozwala na dostęp do tej instancji w przestrzeni globalnej.
- Obiekt ten jest inicjalizowany dopiero wtedy, kiedy jest potrzebny po raz pierwszy.





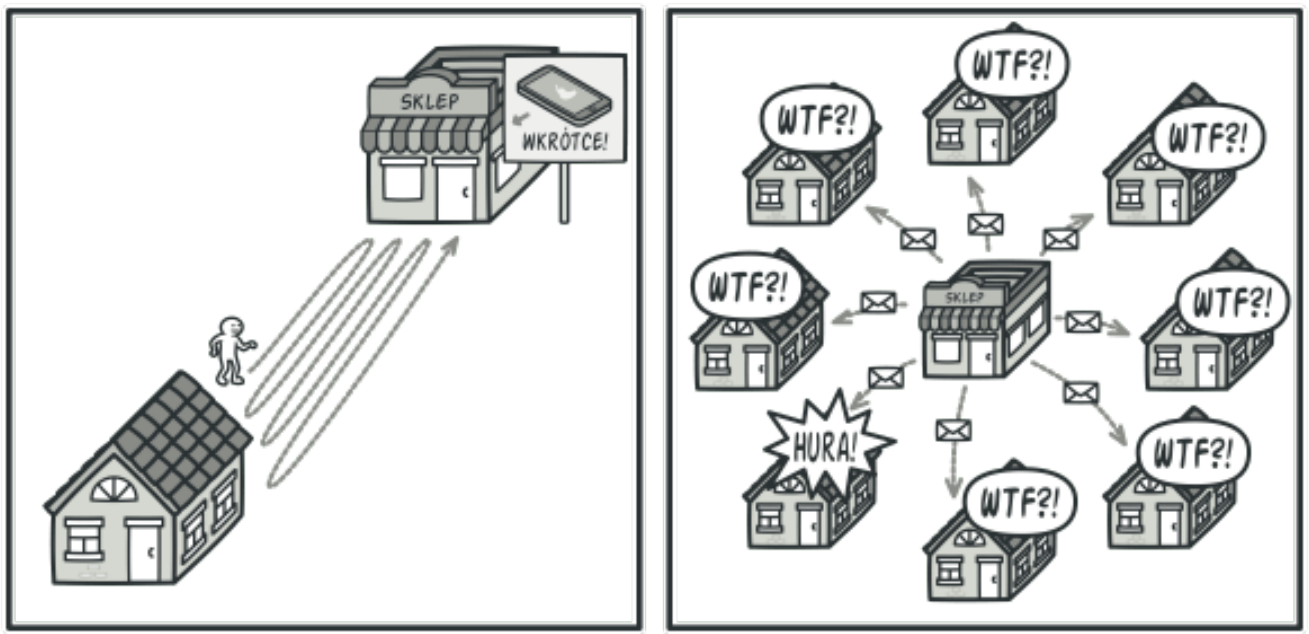
Obserwator (*ang. Observer*)

- Pozwala zdefiniować mechanizm subskrypcji w celu powiadomienia wielu obiektów o zdarzeniach dziejących się w obserwowanym obiekcie.



Obserwator - zalety

- Zasada otwarte/zamknięte. Można wprowadzać do programu nowe klasy subskrybujące bez konieczności zmieniania kodu publikującego (i odwrotnie, jeśli istnieje interfejs publikujący).
- Można utworzyć związek pomiędzy obiektami w trakcie działania programu.



Źródła

- https://infotraining.bitbucket.io/cpp-dp/design-patterns-intro.html
- https://pl.wikipedia.org/wiki/Wzorzec_projektowy_(informatyka)
- https://pl.wikipedia.org/wiki/Adapter_(wzorzec_projektowy)
- https://pl.wikipedia.org/wiki/Fasada_(wzorzec_projektowy)
- https://pl.wikipedia.org/wiki/Obserwator_(wzorzec_projektowy)
- https://pl.wikipedia.org/wiki/Pełnomocnik_(wzorzec_projektowy)
- https://refactoring.guru/design-patterns

Koniec

Dziękuję za uwagę