

Katalog wzorców

Katalog wzorców

- **Wzorce kreacyjne**
- Wzorce behawioralne
- Wzorce strukturalne
- Podsumowanie wzorców GoF

Wzorce kreacyjne

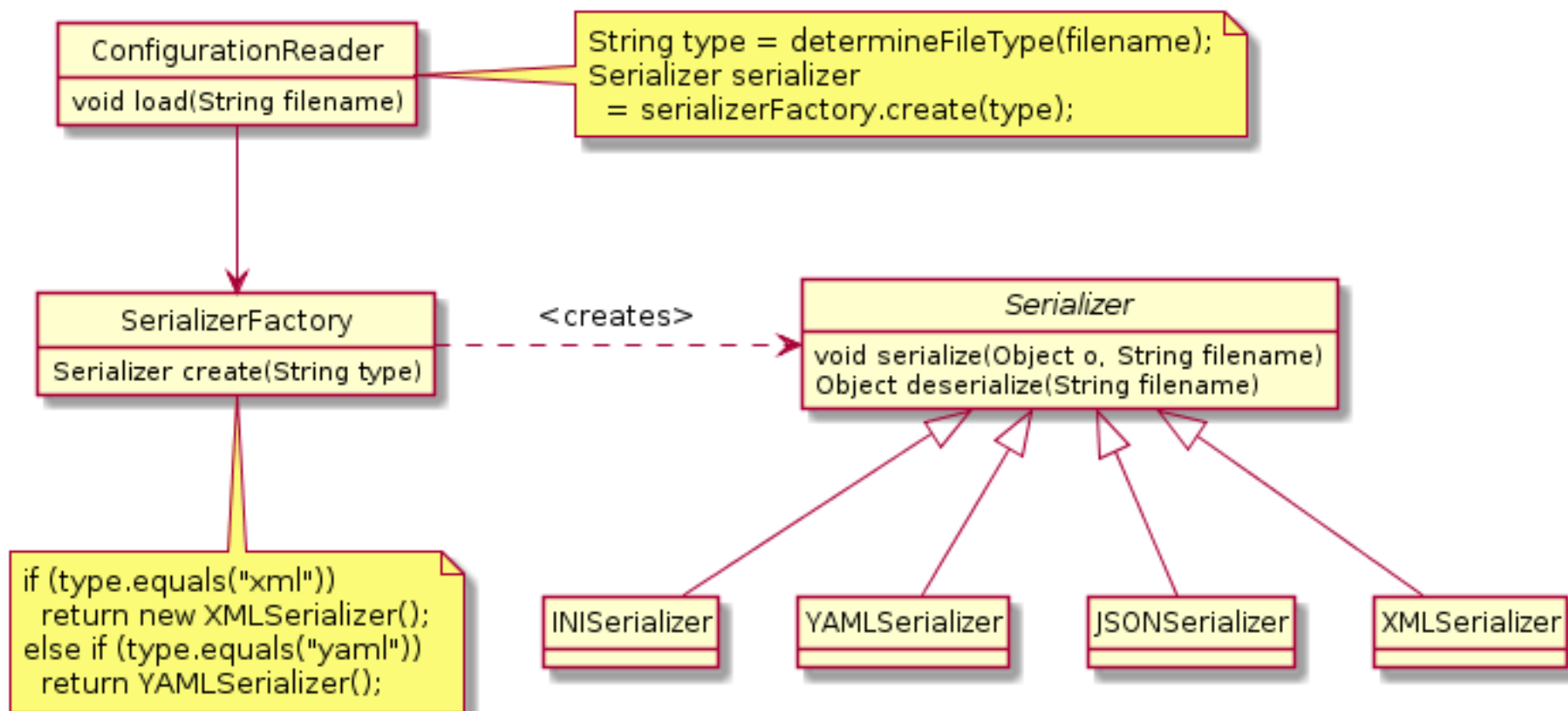
- Prowadzą do utworzenia obiektu.
- Separują tworzenie obiektów od klienta, który je tworzy.
- Ułatwiają budowę systemu, który jest niezależny od sposobu tworzenia i składania obiektów.

Wzorce kreacyjne

- **Simple Factory** - dostarczenie interfejsu do tworzenia obiektów nieokreślonych jako powiązanych typów.
- **Singleton** - ograniczenie możliwości tworzenia obiektów danej klasy do jednej instancji oraz zapewnienie globalnego dostępu do stworzonego obiektu.
- **Builder** - rozdzielenie sposobu tworzenia obiektów od ich reprezentacji. Innymi słowy proces tworzenia obiektu podzielony jest na kilka mniejszych etapów a każdy z tych etapów może być implementowany na wiele sposobów.
- **Prototype** - umożliwienie tworzenia obiektów danej klasy bądź klas z wykorzystaniem już istniejącego obiektu, zwanego prototypem

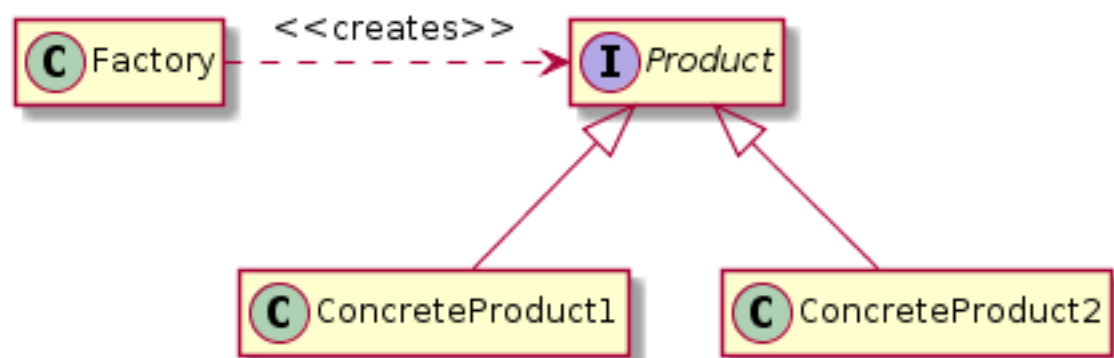
Simple Factory

Przykład



Simple Factory

Struktura



Simple Factory

Cel

- Wzorzec *Simple Factory* hermetyzuje tworzenie rodziny obiektów.
- Podejmuje decyzję o tym jaki obiekt należy utworzyć i tworzy go.

Konsekwencja

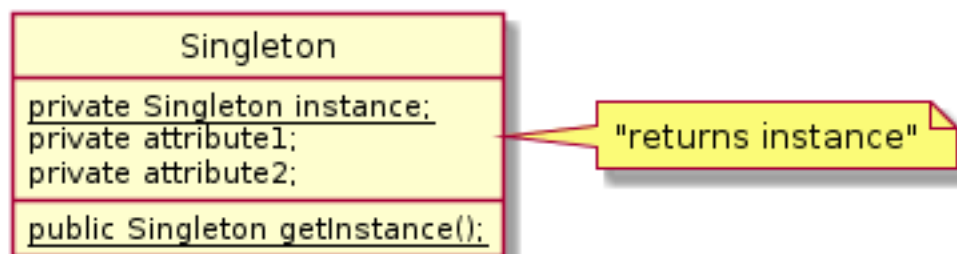
- Pojedyncze miejsce w systemie odpowiedzialne za tworzenie rodziny obiektów

Zastosowanie

- Wyodrębnienie tworzenia obiektów do osobnej dedykowanej ku temu klasy
- Prosta fabryka tworząca obiekt połączenia do bazy danych określonego typu
- Prosta fabryka tworząca obiekt modelu danych na podstawie jego sygnatury

Singleton

Struktura



Singleton

Cel

- Wzorzec Singleton zapewnia, że klasa ma tylko jeden egzemplarz i zapewnia globalny dostęp do niego.

Konsekwencje

- Klasa Singleton może ściśle kontrolować dostęp do swojego jedyne go egzemplarza.
- Można użyć wzorca Singleton do kontrolowania liczby obiektów typu Singleton.
- Wzorzec Singleton wprowadza zmienną globalną i usztywnia projekt.
- Należy unikać stosowania wzorca Singleton, chyba że jest to konieczne.

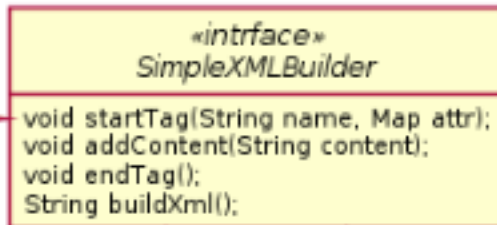
Builder

Przykład

```
SimpleXMLBuilder builder  
= new SimpleXMLBuilder();  
Map attrs = new HashMap();  
attrs.put("id", 1000);
```

```
builder.startTag('person', attrs);  
builder.startTag('name');  
builder.addContent('Jan');  
builder.endTag();  
builder.endTag();
```

```
String xml = builder.buildXml();
```



PrettyXMLBuilder

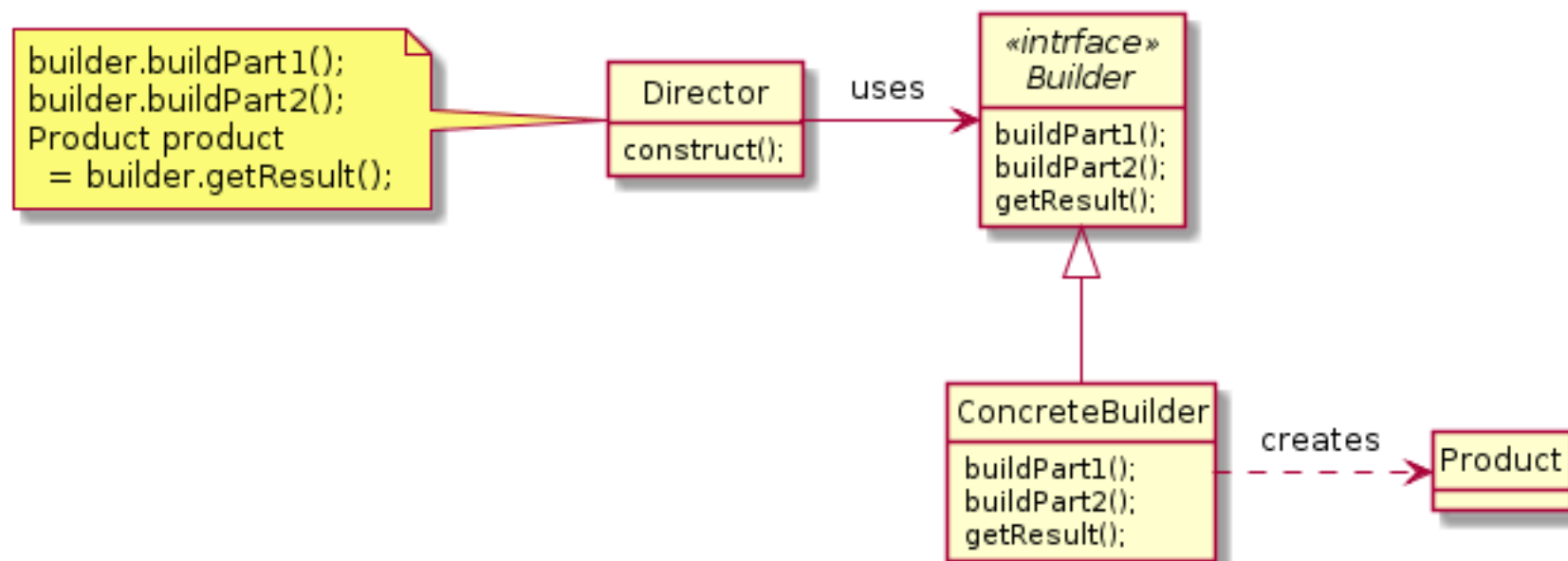
MinifiedXMLBuilder

```
<person id="1000">  
  <name>Jan</name>  
  <surname>Kowalski</surname>  
</person>
```

```
<person id= 1000> <name>Jan</name> <surname>Kowalski</surname> </person>
```

Builder

Struktura



Builder

Cel

- Wzorzec Builder pozwala konstruować obiekty z komponentów.

Konsekwencje

- Wzorzec Builder oddziela kod służący do konstruowania produktu od jego wewnętrznej reprezentacji.
- Klient nie musi nic wiedzieć o wewnętrznej strukturze obiektu Product stworzonego przez obiekt Builder.
- Klient ma wysoką kontrolę nad procesem tworzenia obiektu, gdyż nadzoruje ten proces krok po kroku.

Zastosowanie

- W przypadku gdy potrzebujemy skomplikowaną strukturę obiektową konstruować w prosty sposób.
- Kreator tworzący obiekty w kilku etapach
- Obiekt odpowiedzialny za składanie tekstu i generowanie raportu

Katalog wzorców

- Wzorce kreacyjne
- **Wzorce behawioralne**
- Wzorce strukturalne
- Podsumowanie wzorców GoF

Wzorce behawioralne

- Dotyczą interakcji pomiędzy klasami i obiektami.
- Omawiają sposoby podziału odpowiedzialności pomiędzy klasami.

Wzorce behawioralne

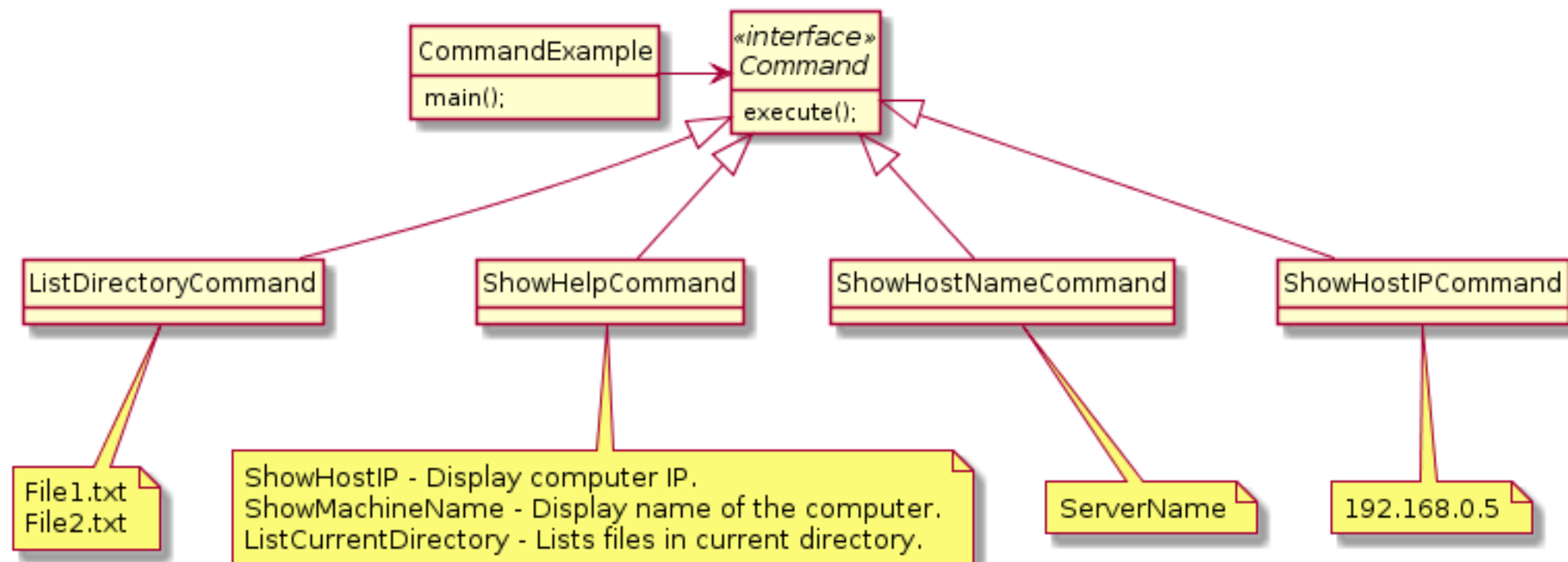
- **Command** - traktujący żądanie wykonania określonej czynności jako obiekt, dzięki czemu mogą być one parametryzowane w zależności od rodzaju odbiorcy, a także umieszczane w kolejkach i dziennikach.
- **Strategy** - definiuje rodzinę wymiennych algorytmów i kapsułkuje je w postaci klas. Umożliwia wymienne stosowanie każdego z nich w trakcie działania aplikacji niezależnie od korzystających z nich klientów.
- **Observer** - używany do powiadamiania zainteresowanych obiektów o zmianie stanu pewnego innego obiektu.
- **Chain of Responsibility** - umożliwia przetwarzanie żądania przez różne obiekty, w zależności od typu

Wzorce behawioralne

- **Template Method** - definiuje metodę, będącą szkieletem algorytmu. Algorytm ten może być następnie dokładnie definiowany w klasach pochodnych.
- **Iterator** - zapewnia sekwencyjny dostęp do podobiektów zgrupowanych w większym obiekcie.
- **State** - umożliwia zmianę zachowania obiektu poprzez zmianę jego stanu wewnętrznego.
- **Visitor** - odseparowanie algorytmu od struktury obiektowej na której operuje. Praktycznym rezultatem tego odseparowania jest możliwość dodawania nowych operacji do aktualnych struktur obiektów bez konieczności ich modyfikacji.
- **Memento** - zapamiętuje i udostępnia na zewnątrz wewnętrzny stan obiektu bez naruszania hermetyzacji. Umożliwia to przywrócenie zapamiętanego stanu obiektu.
- **Mediator** - zapewnia jednolity interfejs do różnych elementów danego podsystemu.

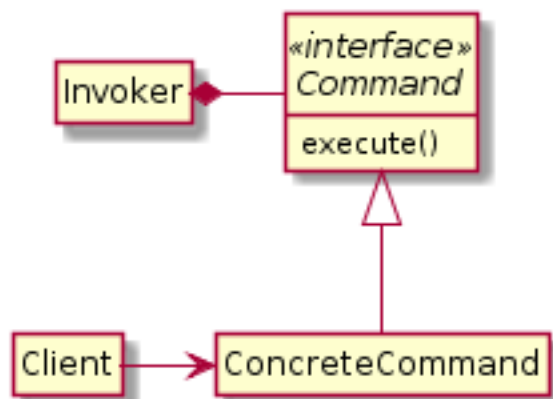
Command

Przykład



Command

Struktura



Command

Cel

- Wzorzec Command hermetyzuje żądania w postaci obiektów.

Konsekwencje

- Wzorzec Command separuje obiekt wywołujący polecenie od obiektu, który wie jak je zrealizować.
- Obiekty Command mogą być rozszerzane tak jak inne obiekty.
- Dodawanie obiektów Command nie wymaga modyfikowania istniejących obiektów klas.

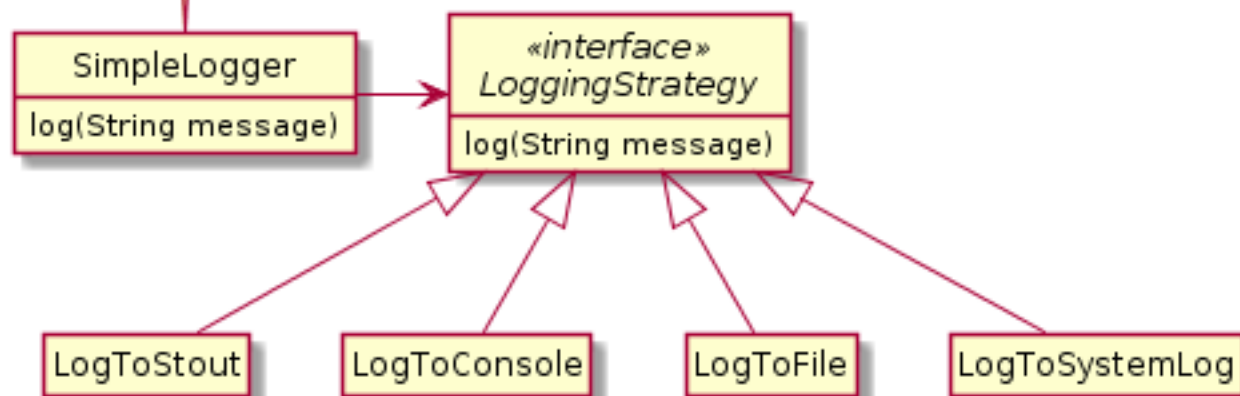
Zastosowanie

- Implementowanie wycofywalnych operacji
 - Kolejkovanie zadań
 - Księgowanie żądań

Strategy

Przykład

```
Logger logger = new SimpleLogger(new LogToFile("file.txt"));  
logger.log("A very important message");
```



Strategy

Cel

- Wzorec Strategy tworzy rodzinę podobnych algorytmów i daje możliwość ich podmiany w trakcie działania programu.

Zastosowanie

- Implementacja jednego algorytmu na różne sposoby
- Serializowanie danych do plików o różnych formatach
- Wyliczanie fragmentu algorytmu na różne sposoby
- Zapisywanie obrazka z wykorzystaniem różnych rodzajów algorytmów

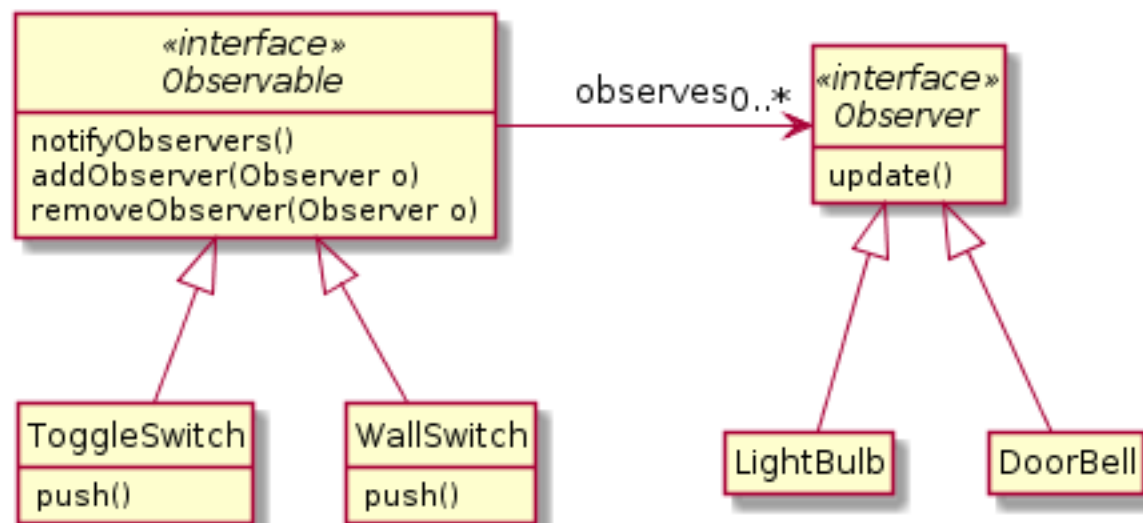
Strategy

Konsekwencje

- Wzorzec Strategy definiuje rodzinę algorytmów powiązanych ze sobą.
- Korzystanie z wzorca Strategy może być alternatywą dla korzystania z dziedziczenia w celu wyodrębnienia nowych algorytmów.
- Hermetyzacja algorytmu w osobnych klasach ConcreteStrategy umożliwia jego modyfikowanie niezależnie od klasy Context.
- Wzorzec Strategy eliminuje przeładowane instrukcje warunkowe.
- Klient musi rozumieć, czym różnią się poszczególne strategie, aby móc dokonać dobrego wyboru strategii.
- Interfejs Strategy jest identyczny dla wszystkich algorytmów, co może prowadzić do przekazywania niepotrzebnych parametrów.
- Nadmierne stosowanie wzorca Strategy może doprowadzić do powstania dużej ilości małych obiektów w systemie.

Observer

Przykład



Observer

Cel

- Wzorzec Observer umożliwia powiadamianie grupy obiektów o zmianie stanu obiektu obserwowanego.

Konsekwencje

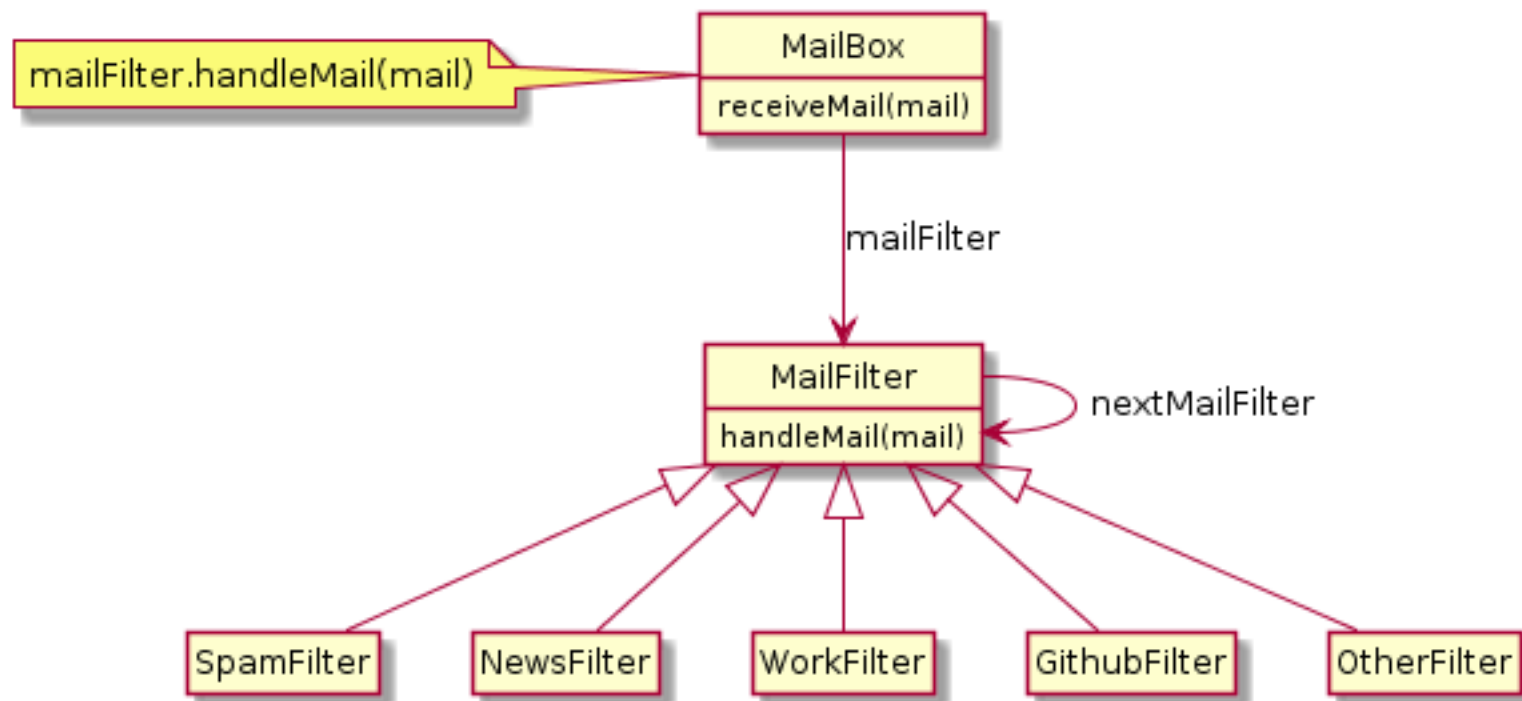
- Wzorzec Observer zapewnia luźne powiązanie pomiędzy obiektami.
- Umożliwia niezależne wymienianie obiektów obserwowanych i obserwatorów.
- Umożliwia dodawanie obserwatorów bez konieczności modyfikowania kodu obiektu obserwowanego.
- Powiadomienie wysyłane przez obserwowanego nie musi specyfikować odbiorcy.
- Prosta operacja może wywołać kaskadę kosztownych uaktualnień.

Zastosowanie

- Informowanie o wydarzeniach zachodzących w systemie
- Informowanie widoku o zmianach w modelu danych

Chain of Responsibility

Przykład



Chain of Responsibility

Cel

- Wzorzec Chain of Responsibility tworzy łańcuch odbiorców i przekazuje wzdłuż niego żądanie, aż jakiś obiekt je obsłuży.
- Separuje nadawcę żądania od jego odbiorców i umożliwia dynamiczne określenie odbiorcy żądania.

Konsekwencje

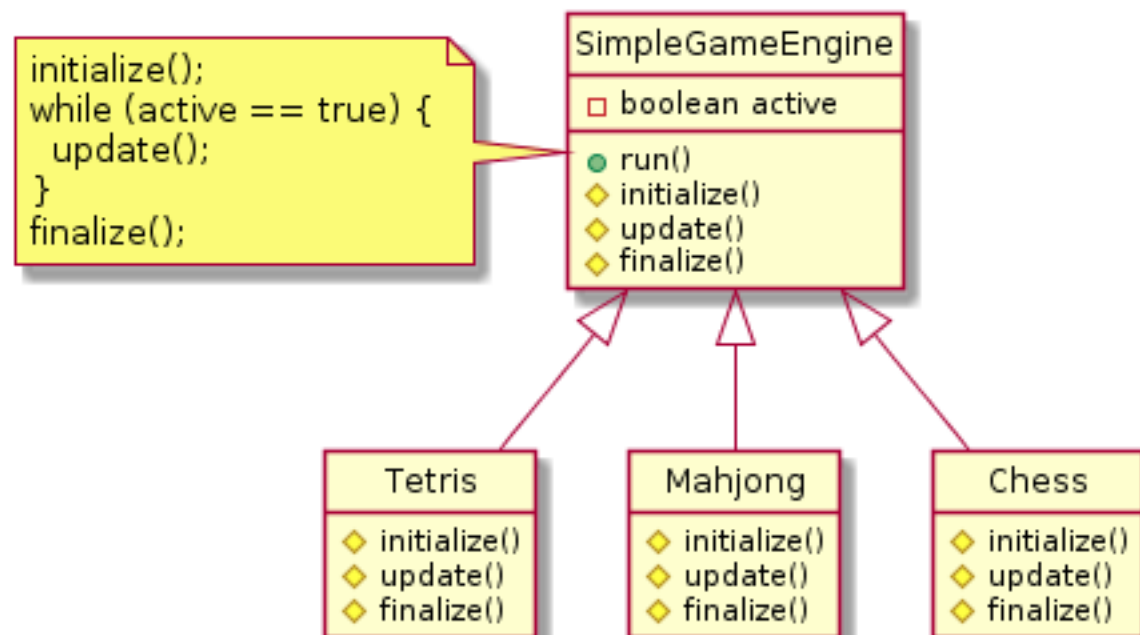
- Nadawca i odbiorca żądania nie są ze sobą powiązani.
- Nadawca żądania nie musi nic wiedzieć o odbiorcy. Wie tylko, że jakiś obiekt je obsłuży.
- Można dynamicznie dodawać/usuwać obiekty obsługujące żądania.
- Istnieje możliwość, że żądanie może zostać nieobsłużone, jeżeli łańcuch jest źle skonfigurowany.

Zastosowanie

- Obsługa zdarzeń myszy i klawiatury w systemach okienkowych.
- Filtrowanie danych w konfigurowalny sposób.

Template Method

Przykład



Template Method

Cel

- Wzorzec Template Method umożliwia podklasom przeddefiniowanie pewnych kroków algorytmu bez zmiany struktury tego algorytmu.

Konsekwencje

- W przypadku implementowania wzorca Template Method ważne jest określenie operacji, które mogą być przeddefiniowywane i tych które muszą być przeddefiniowywane.
- Podklasy mogą rozszerzać działanie operacji z nadklasy poprzez przeciążenie jej i jawne jej wywołanie.
- Nadklasa definiuje wymagane kroki algorytmu i ich kolejność, natomiast podklasy określają, czy chcą te kroki zaimplementować czy też rozszerzyć ich domyślne wersje.

Zastosowanie

- Gotowe szablony do tworzenia dedykowanych rozwiązań.
- Rodziny algorytmów o podobnym ogólnym schemacie działania.
- Gotowy szablon algorytmu, który może występować w różnych odmianach.
- Zastosowanie Template Method do obsługi zapytań bazy danych.

Katalog wzorców

- Wzorce kreacyjne
- Wzorce behawioralne
- **Wzorce strukturalne**
- Podsumowanie wzorców GoF

Wzorce strukturalne

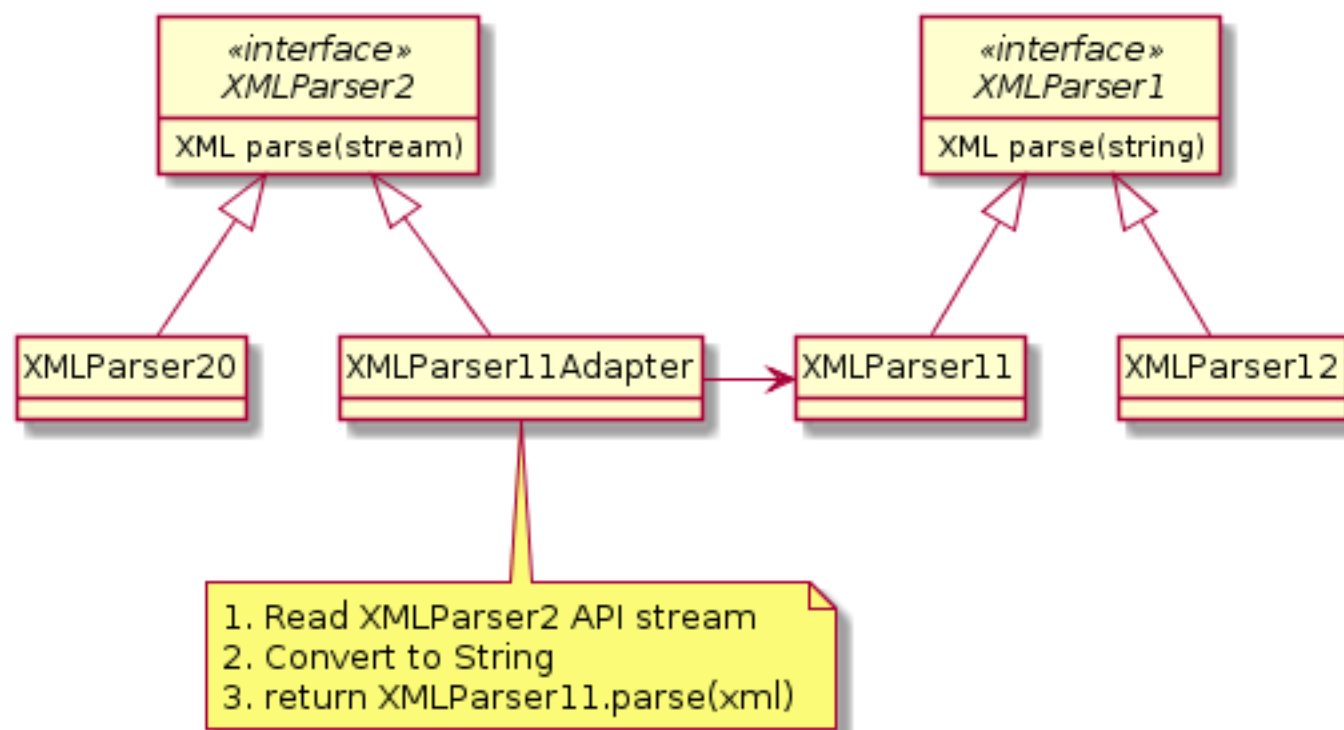
- Opisują sposoby łączenia klas i obiektów w większe struktury.

Wzorce strukturalne

- **Adapter** - umożliwiona współpracę dwóm klasom o niekompatybilnych interfejsach. Adapter przekształca interfejs jednej z klas na interfejs drugiej klasy.
- **Decorator** - polega na opakowaniu oryginalnej klasy w nową klasę "dekorującą". Zwykle przekazuje się oryginalny obiekt jako parametr konstruktora dekoratora, metody dekoratora wywołują metody oryginalnego obiektu i dodatkowo implementują nową funkcję.
- **Facade** - ujednolica dostęp do złożonego systemu poprzez wystawienie uproszczonego, uporządkowanego interfejsu programistycznego, który ułatwia jego użycie.
- **Proxy** - utworzenie obiektu zastępującego inny obiekt. Stosowany jest w celu kontrolowanego tworzenia na żądanie kosztownych obiektów oraz kontroli dostępu do nich.
- **Composite** - składanie obiektów w taki sposób, aby klient widział wiele z nich jako pojedynczy obiekt.

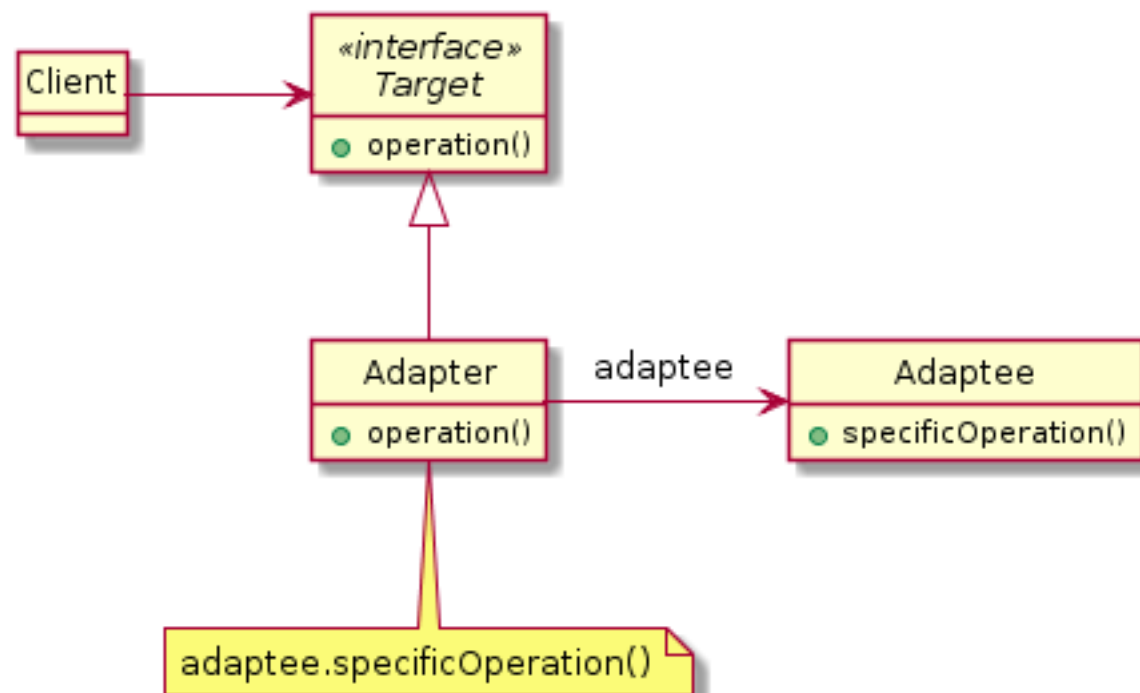
Adapter

Przykład



Adapter

Struktura



Adapter

Cel

- Wzorzec Adapter przekształca interfejs klasy do postaci, której oczekują klienci.

Konsekwencje

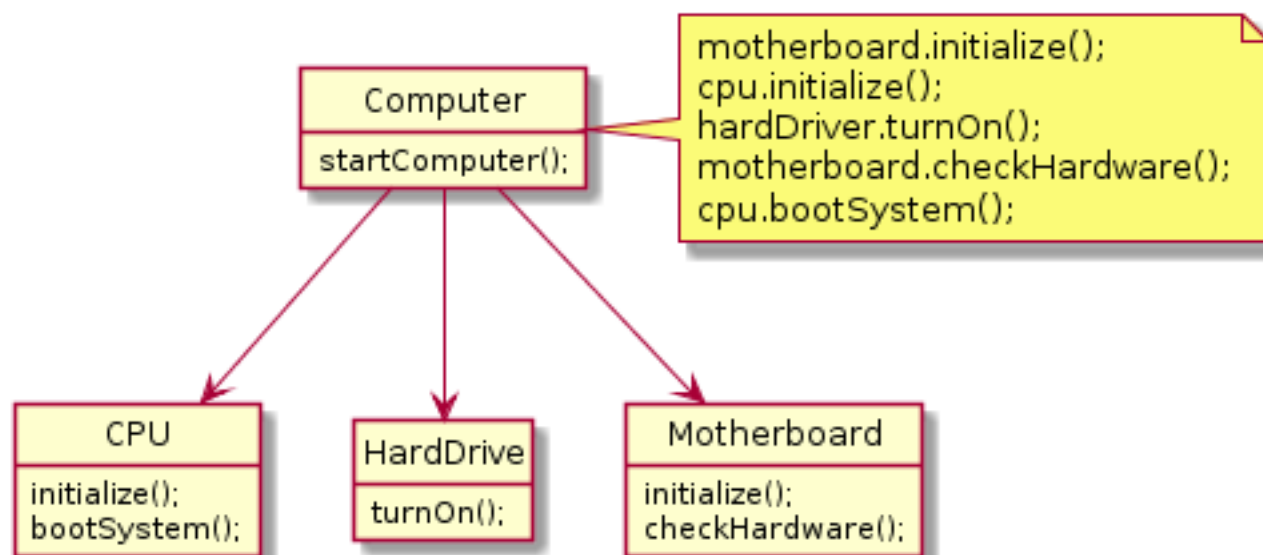
- Adapter może działać również z podklasami obiektu Adaptee.
- Zmiana zachowania obiektu Adaptee wymaga tworzenia podklas i odwoływania się obiektu Adapter bezpośrednio do nich.
- Adapter musi wykonać dodatkową pracę potrzebną do przystosowania interfejsu. Może to pogorszyć wydajność obliczeniową rozwiązania.

Zastosowanie

- Włączanie klasy do systemu, który oczekuje od niej innego interfejsu.
- Adaptowanie zewnętrznej biblioteki do własnych interfejsów.

Facade

Przykład



Facade

Cel

- Wzorzec Facade zapewnia jednolity interfejs dla podsystemu.

Konsekwencje

- Wzorzec Facade zmniejsza ilość obiektów, z którymi klient podsystemu musi współpracować.
- Tworzy słabe powiązanie klienta z podsystemem co umożliwia modyfikowanie podsystemu bez wprowadzania zmian w kliencie.
- Klient może wybrać, czy chce komunikować się z podsystemem za pomocą fasady czy za pomocą bezpośrednich odwołań do jego elementów.

Zastosowanie

- Ułatwienie korzystania ze skomplikowanego podsystemu
- API biblioteki definiujące jej zunifikowany i uproszczony interfejs

Katalog wzorców

- Wzorce kreacyjne
- Wzorce behawioralne
- Wzorce strukturalne
- **Podsumowanie wzorców GoF**

Podsumowanie wzorców GoF

- Jest wiele sposobów implementowania każdego wzorca.
- Diagram dołączany do opisu wzorca jest tylko przykładem a nie specyfikacją.
 - Wzorce są podstawą którą można próbować rozszerzać, parametryzować
- Jest wiele podobieństw pomiędzy wzorcami
- Wzorce występują grupowo, współpracując ze sobą

Różnice i podobieństwa

Strategy

- Hermetyzuje algorytm
- Umożliwia jego dynamiczne podmienianie

Command

- Hermetyzuje żądanie
- Umożliwia wykonanie żądania przez dowolnego klienta
- Umożliwia kolejkovanie żądań i wykonanie ich w późniejszym czasie

Różnice i podobieństwa

Simple Factory

- Tworzy gotowy obiekt
- Potrafi tworzyć różne obiekty na podstawie parametrów

Builder

- Składa gotowy obiekt z części
- Tworzy obiekty podobne różniące się elementami składowymi

Wzorce współpracujące

Strategia

- Obiekty *Strategy* są często produkowane przez obiekty *Factory*
- Wzorzec *Strategy* może korzystać z wzorca *Template Method* w przypadku, gdy algorytm jest rozbudowany
- Obiekt *Strategy* może korzystać ze wzorca *Observer*, aby informować inne obiekty np. o postępie w wykonaniu algorytmu

Wzorce współpracujące

Command

- Wzorzec *Chain of Responsibility* może używać komend do reprezentowania żądań w postaci obiektów.
- Obiekt *Factory* może produkować obiekty *Command*.
- Wzorzec *Decorator* może posłużyć do dynamicznego rozszerzania odpowiedzialności obiektów *Command*.

Antywzorce

Antywzorzec pokazuje jak dojść od problemu do złego rozwiązania.

- pokazuje dlaczego złe rozwiązanie wydaje się korzystne
- pokazuje długofalowe skutki takiego rozwiązania
- wskazuje wzorce, które mogą doprowadzić do dobrego rozwiązania

Antywzorce

Spaghetti Code

Kod staje się nieczytelny na skutek używania złożonych struktur językowych.

The Blob (God Class)

Jedna klasa implementuje zachowanie całej aplikacji, podczas gdy inne przechowują dane.

Golden Hammer

Jedno narzędzie jest używane do rozwiązywania większości problemów.