

Projektowanie obiektowe oprogramowania

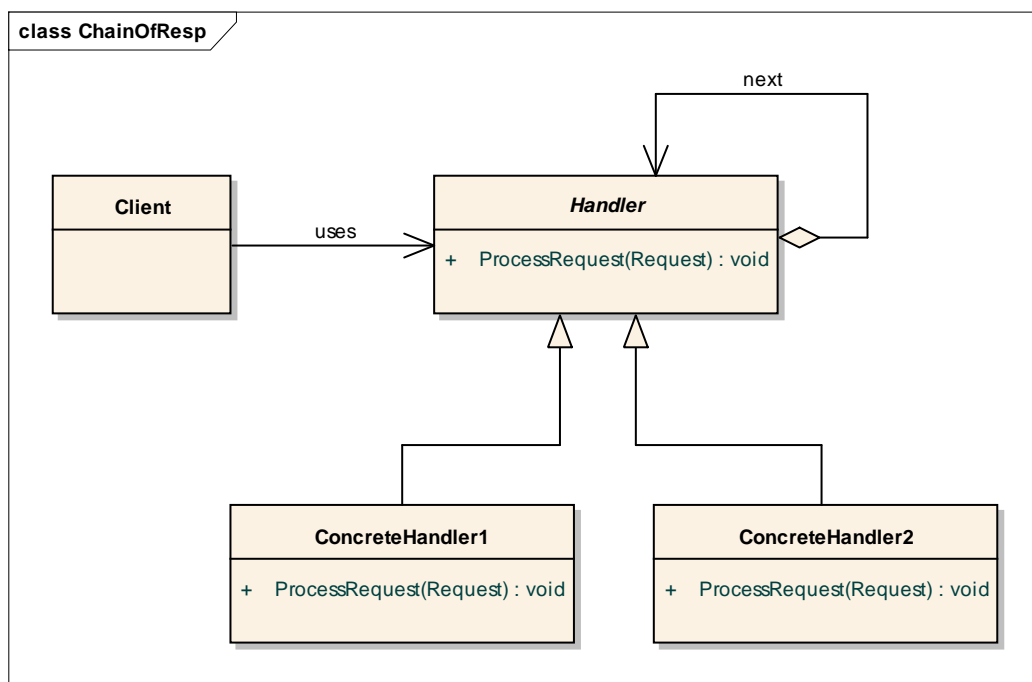
Wykład 8 – wzorce czynnościowe (3)

Wiktor Zychla 2018

1 Chain of Responsibility

Motto: uniknięcie związania obiektu wysyłającego żądanie z konkretnym odbiorcą w sytuacji gdy zbiór możliwych odbiorców jest dynamiczny

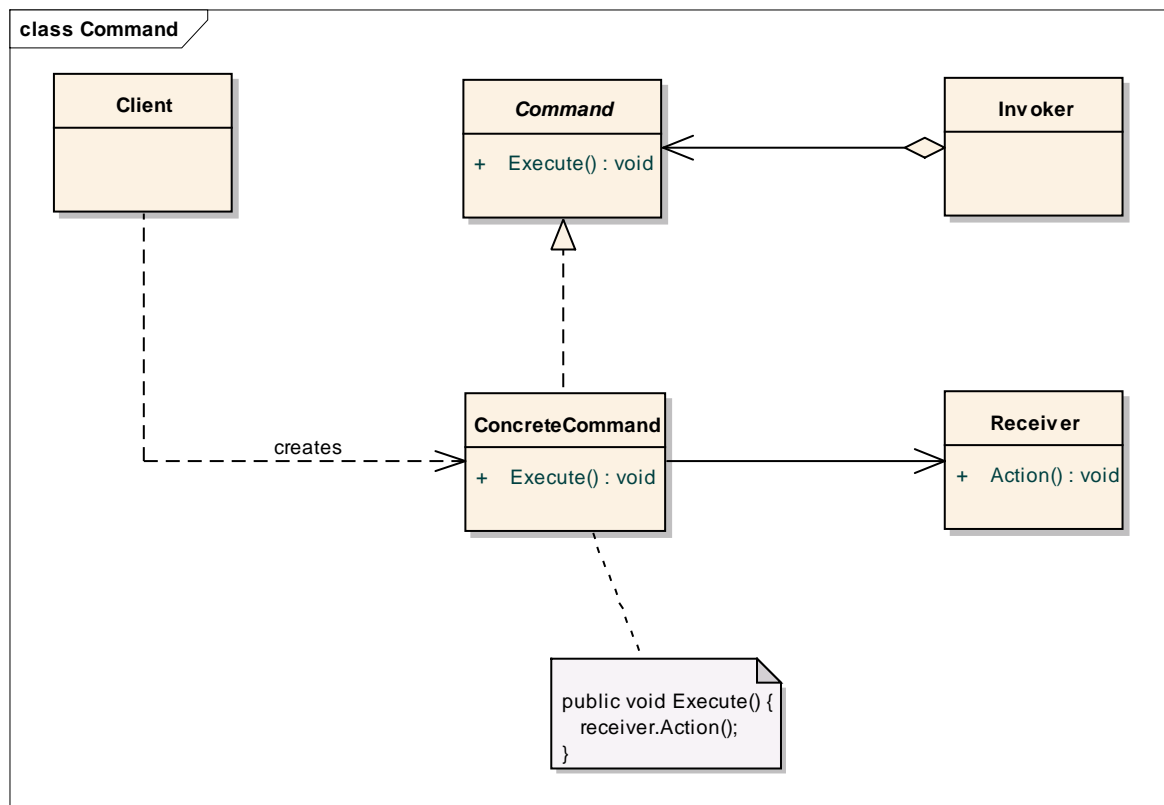
Kojarzyć: łańcuch niezależnych odbiorców wiadomości; przetwarzanie w skomplikowanej logice



2 Command

Motto: kapsułkowanie żądań w postaci obiektów o jednolitym interfejsie dostępu, oddziela wywołującego (Invoker) od odbiorcy (Receiver). Wiele różnych klas Receiver może mieć różny interfejs, ale dzięki Command dla Invokera wyglądają one tak samo. Można powiedzieć, że Command wciela w życie filozofię Adaptera na szeroką skalę – każdy ConcreteCommand jest adapterem jakiegoś Receiver do jednolitego interfejsu Command.

Uwaga: prostszą alternatywą byłoby zapamiętanie funkcji zwrotnej, ale komenda może mieć szerszy interfejs niż tylko Execute(), np. Undo() - o ile sam Receiver może nie implementować Undo wprost, o tyle ConcreteCommand może użyć innego sposobu implementacji Undo (zwykle – z pomocą innego obiektu).



Typowe niedostatki implementacyjne:

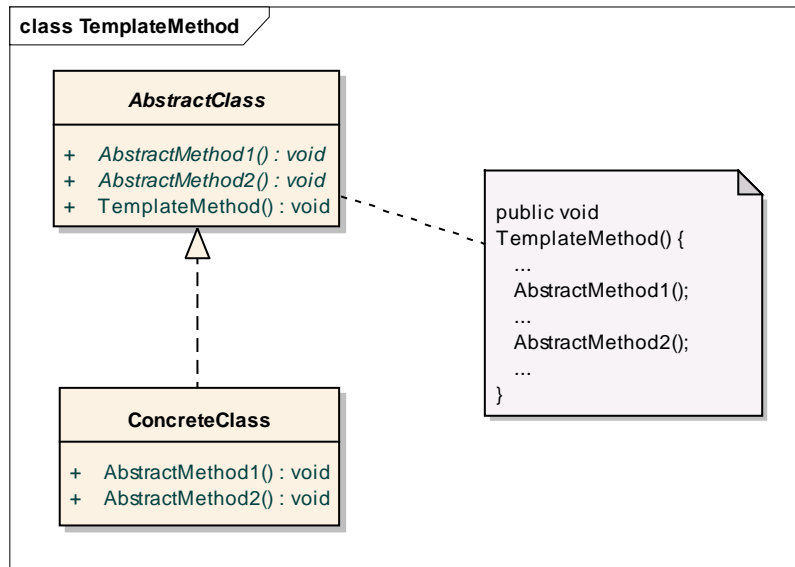
- Konkretnie polecenie nie deleguje wykonania do odbiorcy, tylko całą logikę implementuje samo
- Konkretnie polecenie nie zawiera żadnego stanu, jest tylko delegatorem wywołania do odbiorcy (czy to jest problem ?)

3 Template Method

Motto: określ szkielet algorytmu i zrzucając odpowiedzialność za implementację szczegółów do podklasy.

Kojarzyć z: Template Method = Strategy przez dziedziczenie

Refaktoryzacja Template Method do Strategy polega na zamianie klasy abstrakcyjnej na interfejs i wyłączeniu TemplateMethod do osobnej klasy do której wstrzykiwana jest implementacja interfejsu.

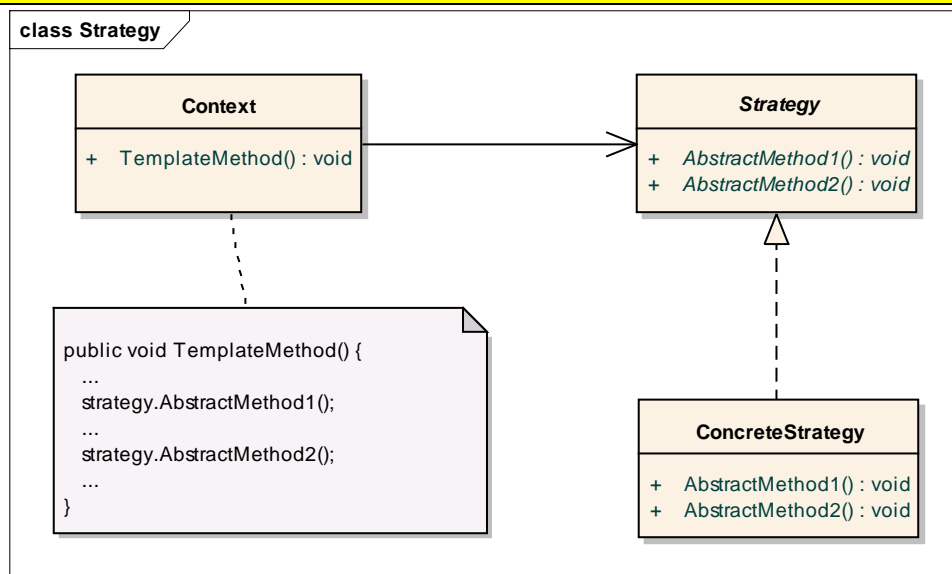


4 Strategy

Motto: określ szkielet algorytmu i zrzucając odpowiedzialność za implementację szczegółów do klasy, do której delegujesz żądania.

Kojarzyć z: Strategy = Template Method przez delegację

Refaktoryzacja Strategy do Template Method polega na likwidacji wstrzykiwania – metoda szablonowa staje się częścią klasy abstrakcyjnej, a konkretną funkcjonalność uzyskuje się przez dziedziczenie.



Template Method vs Strategy

- Template Method wiąże algorytm z konkretną hierarchią klas, Strategy jest implementowane przez interfejs + delegację – można próbować argumentować że Strategy jest lepszym wyborem („*prefer composition over inheritance*”). Z drugiej strony klasa strategii nie ma

zwykle większego sensu poza kontekstem w którym jest używana, a jej struktura jest narzucona przez interfejs, jeśli więc ma wykonać jakąś nietrywialną pracę to to ona *deleguje* tę pracę dalej. W praktyce więc te wzorce

- Template Method może oznaczać mniej kodu, bo klasa przeciąży tylko metody które chce uszczegółwić

5 State

Motto: maszyna stanowa

Uwaga: zbyt rzadko stosowana (można częściej niż się wydaje)

Przykład zastosowania: zarządzanie złożonym GUI. Formularz jest kontekstem, a klasy stanów odpowiadają aktualnie wyświetlanej zawartości. W każdym ze stanów inaczej obsługuje się zestaw zdarzeń formantów.

Przykład:

Zdarzenia:

- klik na drzewie,
- klik na liście,
- dwuklik na liście,
- kliknięcie prawym przyciskiem na liście

Stany

- wyświetlanie listy osób,
- wyświetlanie listy placówek
- wyświetlanie zawartości innych słowników
- itd.

