Daniel Górski
Korporacyjna Java
Wykład 6: Wzorce projektowe

Wzorce projektowe

- Wiele problemów / zagadnień ma wspólne cechy
- Ustandaryzowane rozwiązania są łatwiej przyswajalne dla większego grona odbiorców
- Programiści lepiej się porozumiewają gdy mają wspólny język do opisu zaawansowanych problemów
- W kontekście korporacyjnym większa powtarzalność zadań / problemów daje większe możliwości dokładnego planowania harmonogramu

Wzorzec projektowy

- Nazwa wzorca
- Problem jaki rozwiązuje i zasadność stosowania
- Rozwiązanie i wskazówki implementacyjne w różnych kontekstach
- Konsekwencje zastosowania tego wzorca: wady i zalety

Singleton

- Ograniczenie liczby instancji danej klasy do jednego (lub rzadziej do innej stałej liczby)
  - Daje możliwość posiadania centralnego miejsca odwołań
  - Mamy kontrolę i przewidywalność
  - Często jest określany mianem antywzorca obiektowego
  - W systemach wielowątkowych łatwo aby stał się wąskim gardłem systemu

Builder

- Wsparcie wieloetapowego procesu tworzenia obiektów
  - Duża skalowalność
  - Wydzielenie do wielu wyspecjalizowanych klas cyklu tworzenia złożonego obiektu
  - Można zanadto skomplikować / zwiększyć złożoność niezbyt trudnego problemu

Abstract Factory

- Tworzenie różnych obiektów z tej samej rodziny klas
  - Ukrywa logikę decyzji jakiego dokładnie typu ma być obiekt przed klientem / innym serwisem
  - Dodanie nowego obiektu do rodziny klas wymusza modyfikację w wielu miejscach

Prototype

- Tworzenie obiektu danej klasy na bazie istniejącej już instancji
  - Aktywna wersja: metoda clone() -> obiekt sam tworzy swoją kopię
  - Bierna wersja: konstruktor z wejściem tej samej klasy lub nadklasy -> obiekt jest kopiowany
  - Są dedykowane biblioteki do tego celu, w pewnych kontekstach można to robić samodzielnie
    - Np serializować oryginalny obiekt i budować nowy z powstałego Json'a

Strategy

- Zamknięcie szczegółowej implementacji algorytmu z rodziny algorytmów w postaci klasy
  - Duża elastyczność w możliwości wyboru szczegółowego algorytmu w konkretnym kontekście
  - Ściśle określa formalny interfejs do rozwiązywania danego problemu
  - Zmniejszenie ilości if'ów
  - Dodatkowa warstwa w komunikacji
  - Klasyczny przykład to sortowanie i klasa Comparator: można definiować sposób sortowania nie modyfikując sortowanego obiektu

Iterator

- Zapewnia sekwencyjny dostęp do podobiektów zgrupowanych w większym obiekcie
  - Nie ujawniamy struktury wewnętrznej reprezentowanego zgrupowania
    - Możemy wręcz tworzyć kolejne obiekty na żądanie
  - Możliwość wystawienia różnych porządków sortowania

Flyweight

- Współdzielenie małych niemodyfikowalnych obiektów danego typu
  - Oszczędność na pamięci gdy mamy wiele różnych obiektów o tym samym stanie
  - Przykłady w Javie to String'i deklarowane w kodzie i Integer'y (domyślnie od -128 do 127, można to zmodyfikować w opcjach maszyny wirtualnej)
  - Zmniejszenie wydajności aplikacji gdy użyjemy tego wzorca a ilość różnych obiektów o tym samym stanie jest niewielka

Decorator

- Opakowanie jednej klasy inną aby wzbogacić / rozszerzyć jej działanie
  - Można rozszerzyć działanie klasy nie modyfikując jej
  - Można rozszerzyć działanie klas final
    - String, Double, Long...
    - Klas z zewnętrznych bibliotek
  - Więcej różnych klas
  - Nie można łatwo pracować na kolekcjach oryginalnej klasy i dekoratorów

Proxy

- Stworzenie obiektu zastępującego inny obiekt
  - Virtual proxy: leniwa inicjalizacja kosztownych obiektów
  - Protection proxy: kontrola dostępu do danego obiektu
  - Remote proxy: reprezentacja obiektów z innego serwisu

Command

- Obiekt reprezentujący jakąś akcję do wykonania
  - Można pracować na akcjach do wykonania w obiektowy sposób
    - Można je sortować / priorytezować / łączyć / dzielić...
  - Można stworzyć akcje wycofujące
  - Rozrost struktury klas

Memento

- Zapamiętanie stanu obiektu z jakiegoś momentu w czasie
  - Dobrze współgra z wzorcem Command: łatwo cofnąć stan obiektu do pewnego stanu z przeszłości
  - Zwykle realizowane poprzez tablicę historyczną
  - Potencjalnie duży narzut wydajnościowy jeśli będziemy pracować na obiekcie ciągle modyfikowanym

State

- Uzależnienie działania obiektu od stanu w jakim się znajduje
  - De facto jest to przeniesienie części szczegółowej logiki działania obiektu do kolejnych klas
  - Można dodać kolejny stan i sposób zachowania dla niego nie modyfikując głównego obiektu
  - Raczej nie stosować w przypadku prostych obiektów: powiększenie struktury klas, większa komplikacja niezbyt trudnego problemu

Observer

- Powiadamianie zainteresowanych obiektów o zmianie stanu obiektu obserwowanego
  - Obiekt obserwowany zarządza listą obserwujących
  - Różne możliwości przekazywania informacji co zostało zmienione
    - Bierna: obiekt obserwowany przekazuje referencję do siebie i obserwujący sam sprawdza co się zmieniło
      - Obserwujący musi mieć de facto kopię obiektu obserwowanego
    - Aktywna: obiekt obserwowany przekazuje listę zmienionych właściwości
      - Dodatkowy, często nie mały narzut na stworzenie reprezentacji listy zmian i zarządzanie nią

Visitor

- Oddzielenie algorytmu od struktury obiektowej na której ma on pracować: celem wizytującego jest odwiedzenie wszystkich obiektów w jakiejś strukturze i wykonanie na nich jakiegoś działania
  - De facto jest to wydzielenie pewnego działania na danym obiekcie do zewnętrznego obiektu pracującego na całej grupie obiektów
  - Nie trzeba modyfikować / rozszerzać oryginalnej klasy

Mediator

- Jednolity interfejs do pracy na grupie różnych obiektów
  - Zmniejsza zależności pomiędzy różnymi obiektami poprzez wyekstrahowaniem ich do siebie
  - Obiekty z danej grupy nie komunikują się ze sobą, a delegują zadania do mediatora
  - Przy złożonych strukturach klas jest konieczna jakaś organizacja współpracy...

Pracownia: co wygląda dobrze

- Wiele małych pull-request'ów
  - Mały pull-request łatwiej się przegląda, można w nim łatwiej znaleźć błąd, łatwiej się go wycofuje
  - Bardziej to odzwierciedla typową pracę programisty w korporacji
    - Oczywiście zdarzają się regularnie wyjątki...
- Testy

Pracownia: co wygląda na ciężkie w utrzymaniu

- Długie prace na równoległych branch'ach w repozytorium
  - Branch'e po to są aby móc równoleglić pracę
  - Jeśli oddalą się od siebie za bardzo to trudno potem wszystko scalić

Pracownia

- Od przyszłego tygodnia 1p (w ramach cotygodniowych 10-ciu) będzie przyznawany na podstawie stanu repozytorium
  - Czy prezentacja odbywa się z mastera lub ewentualnie z dosyć świeżego brancha
  - Ilość i wielkość pull-request'ów

Pracownia

- Program ma umożliwić wyświetlenie informacji o wszystkich wprowadzonych figurach posortowanych domyślnie wg pola powierzchni rosnąco
  - W przyszłości dojdą inne sortowania
  - W przyszłości dojdą "masowe" akcje: typu zrób coś ze wszystkimi wprowadzonymi figurami
- Program ma umożliwić rozwiązywanie podstawowych zadań geometrycznych dla trójkąta równoramiennego
  - W przyszłości dojdą kolejne typy trójkątów

Trójkąt równoramienny

- Charakterystyka figury:
  - Długość boku
  - Długość podstawy
  - Pole powierzchni
  - Wysokość (od podstawy)
- Możliwe wejście: (dowolna dwójka)
  - Długość boku
  - Długość podstawy
  - Pole powierzchni
  - Wysokość (od podstawy)