Projektowanie oprogramowania: wzorce architektoniczne i projektowe

Ogólne zasady projektowania

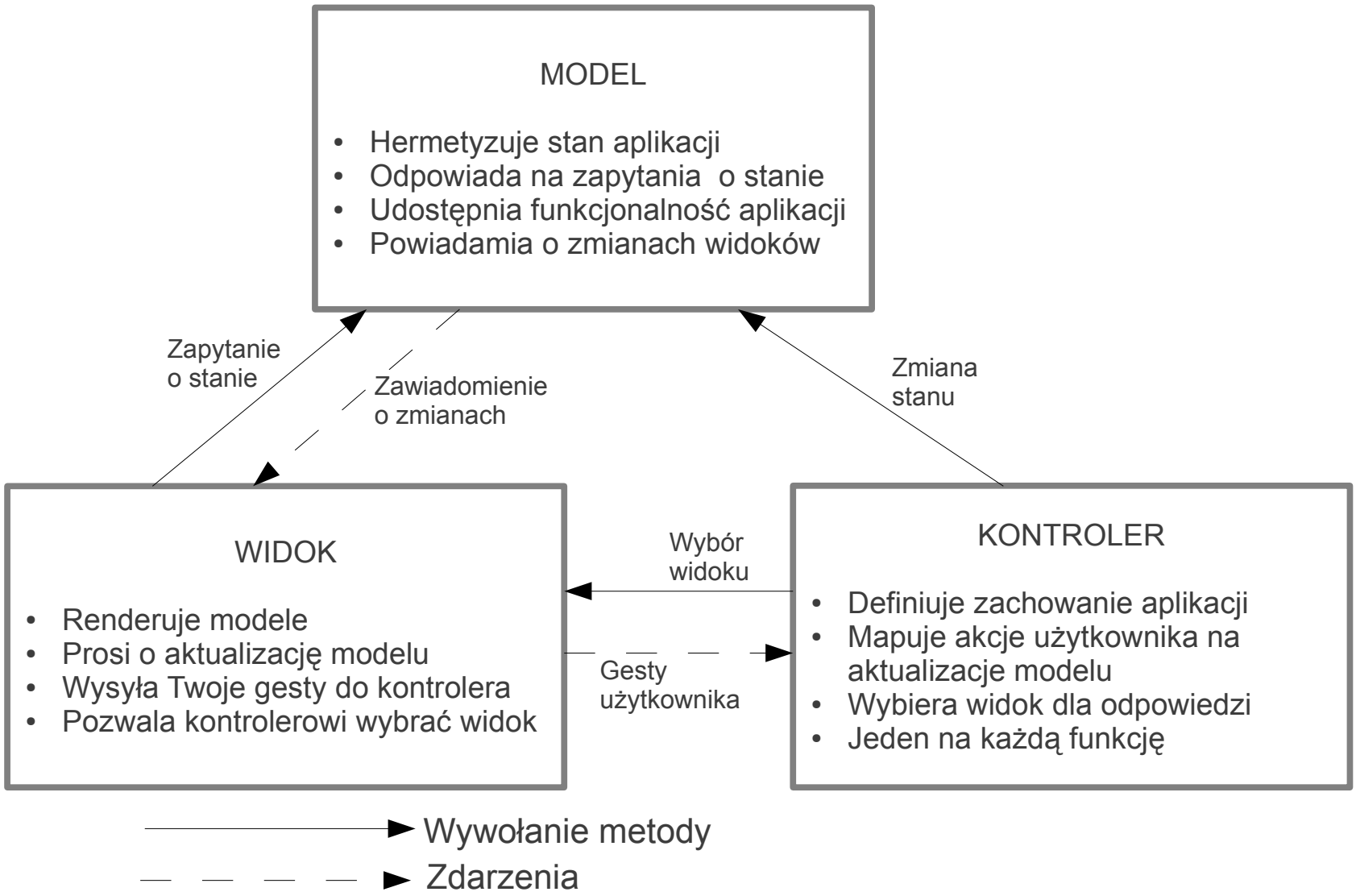
- Nie staraj się zadziwić innych.
- Rzeczy oczywiste rób w sposób oczywisty.
- „Nie rozmawiaj z nieznajomym”.
- Projekt jest skończony, jeżeli już nic więcej nie da się z niego usunąć.
- Prostota przed ogólnością.
- „Izomorfizm”: jeden obiekt abstrakcyjny – jedna klasa.
- Unikaj powielania kodu.

Wzorce architektoniczne

- „Każdy wzorzec opisuje problem, który ciągle na nowo pojawia się w naszym otoczeniu i opisuje rdzeń jego rozwiązania w taki sposób, że można go używać milion razy i nigdy w ten sam sposób.”

(Christopher Alexander)

Wzorzec MVC

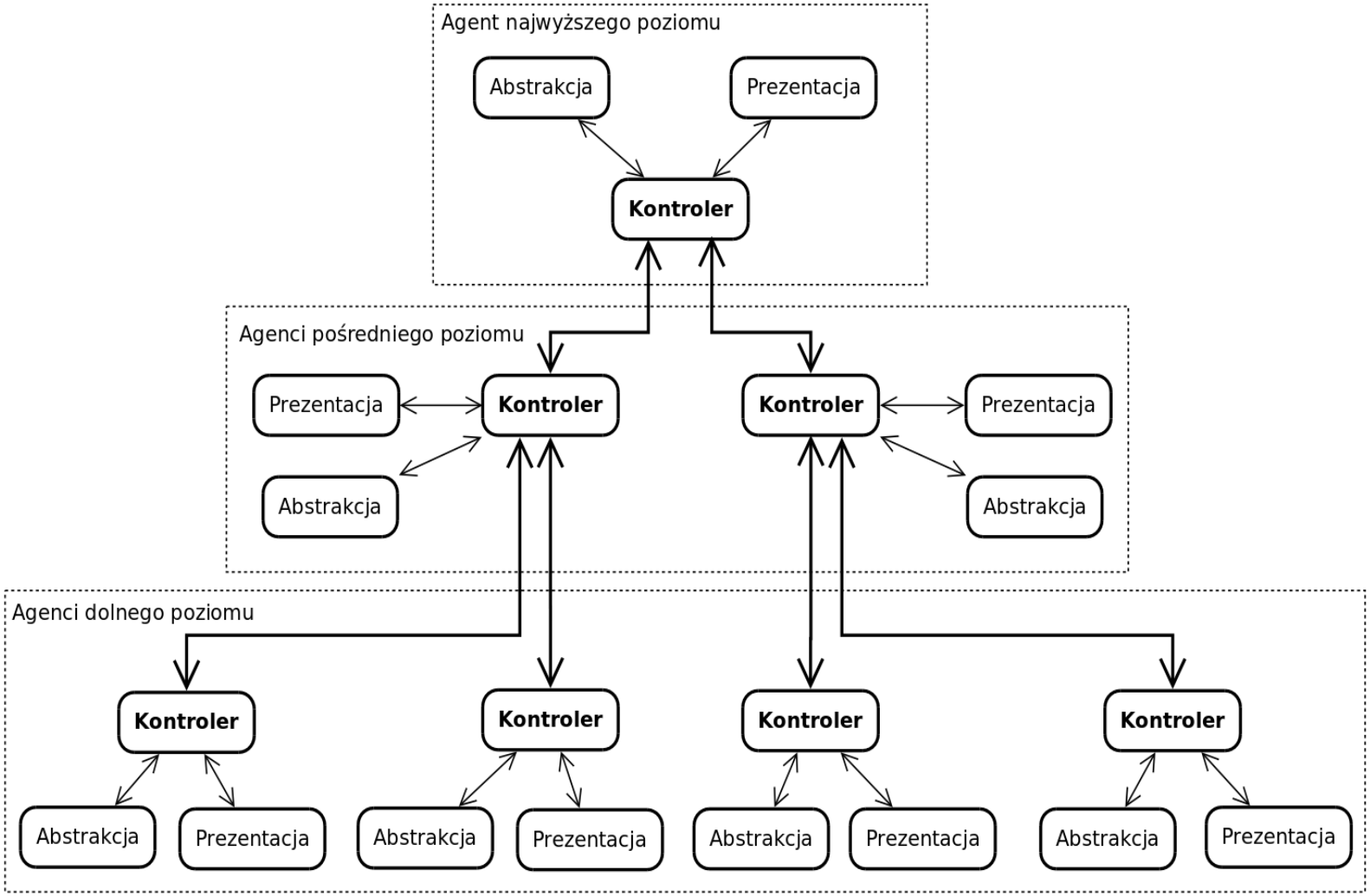

MODEL
Hermetyzuje stan aplikacji
Odpowiada na zapytania o stanie
Udostępnia funkcjonalność aplikacji
Powiadamia o zmianach widoków
Zapytanie o stanie
Zawiadomienie o zmianach
Zmiana stanu
WIDOK
Renderuje modele
Prosi o aktualizację modelu
Wysyła Twoje gesty do kontrolera
Pozwala kontrolerowi wybrać widok
Wybór widoku
Gesty użytkownika
KONTROLER
Definiuje zachowanie aplikacji
Mapuje akcje użytkownika na aktualizacje modelu
Wybiera widok dla odpowiedzi
Jeden na każdą funkcję
Wywołanie metody
Zdarzenia

Wzorzec MVC, cd.

- Na bazie wzorca MVC powstało wiele wzorców pochodnych o zmienionej strukturze oraz właściwościach. Jeden z nich to wzorzec PAC (ang. Presentation – abstraction – control, PAC) opisuje architekturę oprogramowania zorientowaną na interakcję i jest używany jako hierarchiczna struktura agentów, w której każdy agent składa się z triady prezentacji, abstrakcji i kontroli.

- Agenci komunikują się tylko przez element kontroli. W odróżnieniu od MVC w każdej triadzie całkowicie izoluje się prezentację (widok w MVC) i abstrakcję (model MVC), zapewnia to możliwość oddzielnej wielowątkowości modelu i widoku, co, np. może dać użytkownikowi wrażenie bardzo krótkiego czasu startu aplikacji, bo interfejs użytkownika (prezentacja) może być wyświetlana przed pełną inicjalizacją abstrakcji.

Wzorzec PAC


Agent najwyższego poziomu
Abstrakcja
Prezentacja
Kontroler
Agenci pośredniego poziomu
Prezentacja
Kontroler
Kontroler
Prezentacja
Abstrakcja
Abstrakcja
Agenci dolnego poziomu
Kontroler
Kontroler
Kontroler
Kontroler
Abstrakcja
Prezentacja
Abstrakcja
Prezentacja
Abstrakcja
Prezentacja
Abstrakcja
Prezentacja

Wzorce projektowe (szablony)

Wzorzec projektowy w informatyce jest:

- „Opisem komunikujących się obiektów i klas, które są specjalnie wykonane, aby rozwiązać ogólny problem przy dokładnie określonym kontekście”.

(GangOfFour, 1994)

- „Powracającym rozwiązaniem zagadnień projektowych, z którymi wciąż się spotykamy”.

(Alpert, 1994)

- „Zbiorem reguł opisujących, w jaki sposób osiągnąć pewne cele w dziedzinie programowania”.

(Pree, 1994)

Wzorce projektowe, cd.

Elementy wzorca projektowego:

- Nazwa wzorca - odzwierciedla problem, rozwiązanie i konsekwencje danego wzorca.
- Problem - opisuje zagadnienie i kontekst wystąpienia wzorca.
- Rozwiązanie - opisuje elementy tworzące projekt, ich relacje, odpowiedzialności oraz współpracę.
- Konsekwencje – są to rezultaty zastosowania wzorca (korzyści i straty).

Klasyfikacja wzorców projektowych

- Konstrukcyjne (Creational) - dotyczą procesu tworzenia obiektu.
- Strukturalne (Structural) - dotyczą struktury klas lub obiektów.
- Behawioralne (Behavioral) - charakteryzują sposób, w jaki klasy lub obiekty oddziałują i dzielą odpowiedzialności.

Katalog wzorców projektowych

		Przeznaczenie		
		Konstrukcyjne (Creational)	**Strukturalne (Structural)**	**Behawioralne (Behavioral)**
Zakres	**Klasa**	Factory Method	Adapter	Interpreter Template Method
	Obiekt	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

Opis wzorca projektowego

- Nazwa wzorca i klasyfikacja - krótko ukazuje istotę wzorca.
- Intencja – pokazuje co dany wzorzec robi, do jakiego zagadnienia lub problemu się odnosi.
- Motywacja - ukazuje problem i sposób, w jaki struktury klas i obiektów go rozwiązują.
- Stosowalność – określa w jakich sytuacjach dany wzorzec może być wykorzystany.

Opis wzorca projektowego c.d.

- Struktura – jest to graficzna reprezentacja klas we wzorcu.
- Elementy – są to klasy i obiekty uczestniczące w strukturze wzorca, oraz zakres ich odpowiedzialności.
- Powiązania – pokazują jak powiązane są elementy struktury.
- Konsekwencje – opisują jakie są skutki użycia danego wzorca.

Opis wzorca projektowego c.d.

- Implementacja – są to ogólne wskazówki i techniki programowania, o których powinniśmy wiedzieć. Istotne jest uwzględnienie specyfiki języka.
- Przykładowy kod - fragmenty kodu ilustrujące przykładową implementację.

Wzorce konstrukcyjne (Creational patterns)

Fabryka (Factory Method)
Fabryka Abstrakcyjna (Abstract Factory)
Singleton
Prototyp (Prototype)
Budowniczy (Builder)

Wzorce konstrukcyjne

- Zajmują się tworzeniem instancji obiektów
- Uniezależniają program od obiektów, które tworzą
  - Obudowują wiedzę na temat konkretnych klas, które nasz system używa
  - Ukrywają sposób, w jaki instancje tych klas są kreowane
- Pozwalają na wprowadzenie większej elastyczności do kodu naszego programu

Fabryka (Factory Method)

- Intencja
  - Wzorzec ten, w zależności od dostarczonych danych, zwraca instancję jednej z możliwych klas
  - Wszystkie zwracane klasy mają zazwyczaj wspólnego przodka i wspólne metody, jednak każda z nich wykonuje swoje zadania w inny sposób
- Stosowalność
  - Gdy klasa nie może przewidzieć, jakiej klasy obiekty powinna stworzyć
  - Gdy wpływ na rodzaj tworzonego obiektu mają dane

Przykład: Fabryka (Factory Method)

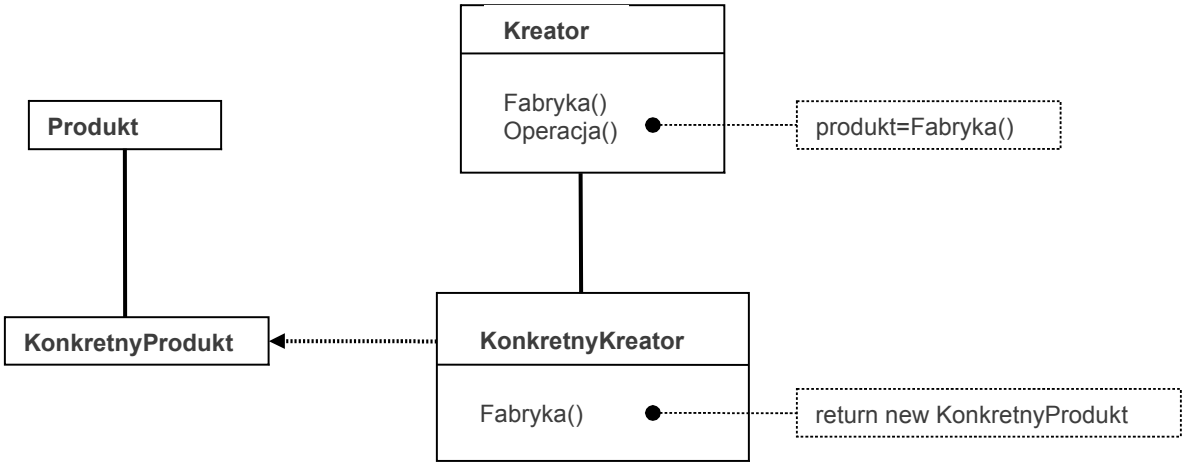
- Intencja
  - Wzorzec ten, w zależności od dostarczonych danych, zwraca instancję jednej z możliwych klas
  - Wszystkie zwracane klasy mają zazwyczaj wspólnego przodka i wspólne metody, jednak każda z nich wykonuje swoje zadania w inny sposób
- Stosowalność
  - Gdy klasa nie może przewidzieć, jakiej klasy obiekty powinna stworzyć
  - Gdy wpływ na rodzaj tworzonego obiektu mają dane

Fabryka – przykładowy kod

```
public class Osoba {
        public String plec; //K (kobieta) lub M (mężczyzna)
        public String jakaPlec() { return plec; }
}
public class Kobieta extends Osoba {
        public Kobieta() { System.out.println(„Dzien dobry Pani”); }
}
public class Mezczyzna extends Osoba {
        public Mezczyzna() { System.out.println(„Dzien dobry Panu”); }
}
public class Powitanie {
        public static void main(String[] args) {
        Powitanie dziendobry=new Powitanie();
        dziendobry.kto(arg[0]);
        }
        public Osoba kto(String plec) {
        if (plec.equals(”K”)) return new Kobieta();
        if (plec.equals(”M”)) return new Mezczyzna();
        else return null;
        }
}
```

Fabryka – struktura i konsekwencje

- Struktura



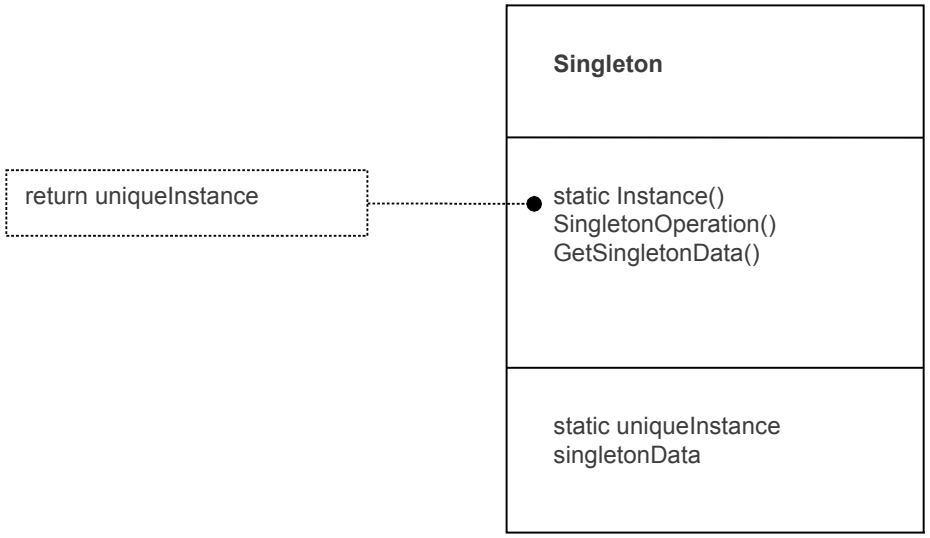
- Konsekwencje
  - Tworzenie obiektów w klasie z fabryką jest bardziej elastyczne niż tworzenie ich bezpośrednio
  - Łączy logicznie klasy opisujące pojęcia równoległe
  - Wada: klient musi wytworzyć podklasę kreatora (KonkretnyKreator) czasami tylko po to, żeby uzyskać KonkretnyProdukt

Przykład: Singleton

- Intencja
  - Zapewnia, że dana klasa w każdym momencie ma tylko jedną instancję i udostępnia ją globalnie
- Motywacja
  - Singleton polega na stworzeniu klasy, która przechwytuje żądania powstania liczniejszych instancji, oraz zapewnia łatwy dostęp do tej jedynej istniejącej
  - Jedyna instancja jest rozszerzalna dla jej podklas, w których można jej używać bez modyfikacji kodu
  - Przykłady użycia: w systemie może występować tylko jeden manager okien, jedyny punkt dostępu do baz danych, jedna kolejka do jednej drukarki

Singleton – struktura, konsekwencje

- Struktura
  - Operacja Instance pozwala
  - klientowi dostać się do
  - jedynej instancji (uniqueInstance)
  - Singleton może być odpowiedzialny
  - za tworzenie swojej własnej unikalnej instancji
- Konsekwencje
  - Kontrolowany dostęp do jedynej instancji
  - Zredukowana przestrzeń nazw
  - Pozwala na zwiększenie dozwolonej ilości instancji



Singleton - przykładowy kod

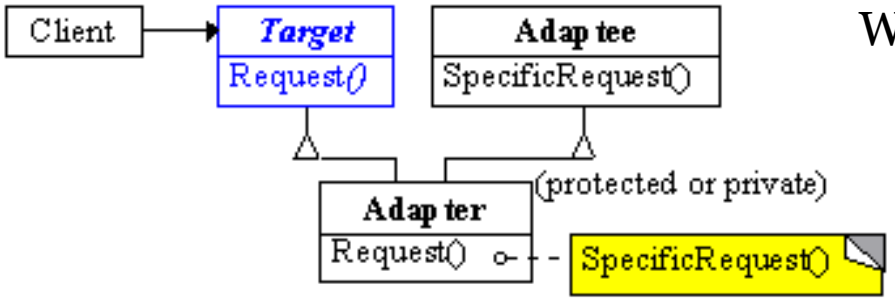
```
public class Singleton {

    /*tworzymy konstruktor protected; gdyby był private chroniłby
     *przed tworzeniem instancji Singletonu, jednak byłoby
niemożliwe
     *tworzenie instancji jego podklas
    */

    protected Singleton () {

    // ...

    }

    static public Singleton _instance = null;

    /*metoda instance() zwraca jedyną instancję Singletonu;
     *użycie: Singleton.instance(
     */

    static public Singleton instance() {

    if (null == _instance) {

    _instance = new Singleton();

    }

    return _instance;

    }

}
```

Wzorce strukturalne

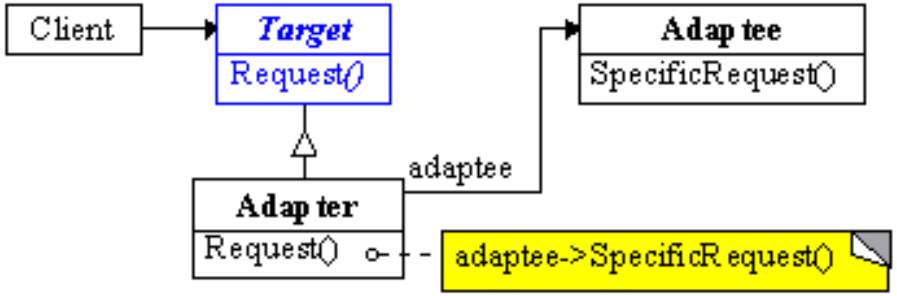
Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy

Przykład: Adapter

Dopasowanie (adaptowanie) interfejsu klasy.
Do użycia, gdy mamy istniejącą klasę której chcemy użyć, ale nie zgadza się interfejs.



Wersja z wielo-dziedziczeniem



Wersja z delegacją

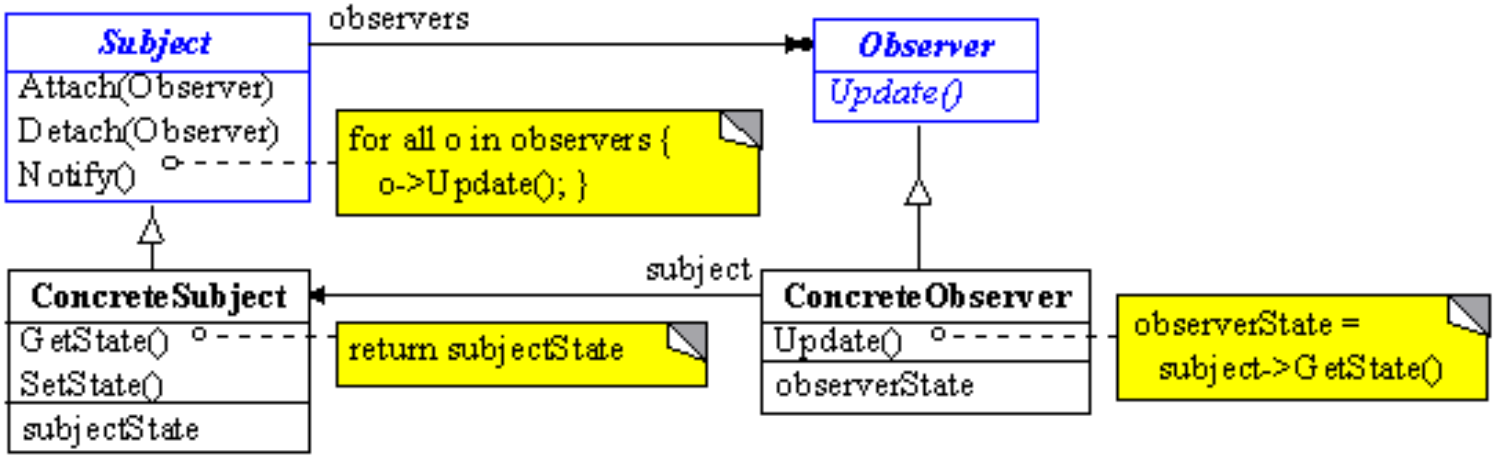
- **Bridge** - podobny do wzorca Adapter. Osobna klasa dla abstrakcji i osobna dla implementacji. Dwie hierarchie dziedziczenia. Implementacja jest ukrywana.
- **Composite** - sugeruje aby tworzyć drzewiaste struktury w taki sposób, że węzły i liście mają ten sam interfejs.
- **Decorator** - omawiany wcześniej
- **Facade** - stosowany gdy mamy złożony zestaw obiektów ze skomplikowanymi zależnościami. Dodajemy nowy obiekt który dostarcza prostego interfejsu. Ogranicza to możliwości, ale zawsze można skorzystać z pełnego zestawu obiektów.
- **Flyweight** - gdy występuje potrzeba użycia bardzo wielu instancji danej klasy. Wzorzec ten pokazuje jak tworzyć jedną instancję zastępującą wiele.
- **Proxy** - pośrednik. Działania wykonywane są na nim, a on działa na właściwej klasie (o tym samym interfejsie). Szczególnie użyteczny w środowisku sieciowym.

Wzorce czynnościowe

State, Observer, Chain of Responsibility, Command, Iterator, Mediator, Strategy, Visitor

Przykład: Observer

- Gdy jeden z obiektów (*Observer)* chce być informowany o zmianach zachodzących w innym obiekcie (*Subject)*.
- Może być kilka obserwatorów jednego obiektu klasy *Subject*.
- Obiekty w tej relacji mogą być luźno powiązane.



Chain of Responsibility - nadawca wysyła żądanie do jednego z obiektów, a obiekty z łańcuch przekazują sobie żądanie.

Command - żądanie ma formę obiektu

Iterator - poruszanie się po dowolnej kolekcji danych

Mediator - pośrednik między obiektami - brak jawnych odwołań

Strategy - podobnie do State. Różnica jest taka, że w State zmiany zależały od zmian w wewnętrznym stanie obiektu - tutaj zmiana jest wykonywana z zewnątrz. Dodatkowo w State każda z klas mogła robić zupełnie co innego, a tutaj ma robić dokładnie to samo (ale być może w inny sposób).

Visitor - klasa *Visitor* odwiedza każdy z obiektów.

Źródła wiedzy

1. Cooper J. W. , Java. Wzorce Projektowe, Gliwice 2000

2. Shalloway A., Projektowanie zorientowane obiektowo. Wzorce projektowe. Wyd. Helion 2005

3. Christopher Alexander,An Introduction for Object-Oriented Designers, http://g.oswego.edu/dl/ca/ca/ca.html

4. Kremer R., Programming Patterns Overview, http://pages.cpsc.ucalgary.ca/~kremer/patterns/

5. Walter B., Wzorce projektowe, http://wazniak.mimuw.edu.pl/images/8/80/Io-8-wyk.pdf