Podsumowanie wzorców projektowych GoF

Wielowymiarowość wzorców
Przykłady z życia w Java
Case-study z książki GoF w Java

Wielowymiarowość wzorców

Zagadnienie wzorców projektowych jest znacznie bardziej skomplikowane niż można by sądzić na podstawie samego wykładu. Na wykładzie omówiono bowiem jedynie podstawowe wersje wzorców, a mianowicie wzorce w wersji z klasami abstrakcyjnymi, jednowątkowe,... Jak wynika zarówno z powyższego stwierdzenia jak i z samej natury zagadnienia wzorce projektowe mogą mieć rozliczne warianty. Warianty te wynikają z przynależności wzorców projektowych do następującej przestrzeni:

$$DP = (R, T, W, D)$$

gdzie:
R - realizacja (klasy abstrakcyjne, interfejsy, klasy wewnętrzne,...)
T - technologia (język programowania, platforma implementacyjna,...)
W - jedno lub wielowątkowość
D - dziedzina zastosowań (ogólnego przeznaczenia, telekomunikacyjne,...)

Na zakończenie tego ogólnego posumowania warto przedstawić następujące uwagi:
UWAGA 1:
W dalszych częściach niniejszej prezentacji ograniczymy się z konieczności jedynie do przykładów z podprzestrzeni:
Java, single-threaded, abstract/interface, ...

UWAGA 2:
Warto też zwrócić przy okazji uwagę, że prezentowane na wykładzie oraz w tej prezentacji wzorce projektowe są wzorcami niskopoziomowymi, a więc odwołującymi się jedynie do poziomu języka programowania. Istnieje cały szereg innych wzorców projektowych odwołujących się do poziomu różnych platform wykonawczych, np. J2EE, SOA,...

UWAGA 3:
Wzorce projektowe GoF należy traktować jako początkowy zbiór wzorców niskopoziomowych. Istnieją również inne wzorce opracowane później lub niezależnie od prezentowanych wzorców GoF.

UWAGA 4:
Przezentowane wzorce GoF to wzorce ogólnego przeznaczenia. Istnieją jednak również wzorce projektowe specyficzne dla różnych dziedzin zastosowań, jak np. dla systemów czasu rzeczywistego, telekomunikacji i innych.

UWAGA 5:
Wzorce projektowe często nie występują pojedynczo, lecz są zestawiane z wielu współpracujących ze sobą wzorców. Zagadnienie to jest trudniejsze od używania wzorców projektowych pojedynczo. Omówienie w pełni tego zagadnienia przekracza zakres zajęć ze względu na ogromną różnorodność powiązań wzorców.

UWAGA 6:
Tak jak istnieją wzorce projektowe pokazujące dobre praktyki programowania tak istnieją tzw. anti-patterns pokazujące najczęstsze błędy jakie popełnia się przy tworzeniu kodu źródłowego oprogramowania oraz omawiające konsekwencje tych błędów.

Problem znalezienia lub opracowania rozlicznych wariantów wzorców wynikających z powyższych uwag pozostawiam czytelnikowi.

Przesłanki do stosowania wzorców projektowych:

1. Wychwycenie tego, co się zmienia i zakapsułkowanie tego w obiekcie z hiererachii klas.
2. Wymuszenie na programiście wprowadzającym zmiany spełnienia narzuconych ograniczeń lub bycia w zgodzie z narzuconymi konwencjami poprzez mechanizm kompilacji.

Przykłady w Java

W czasie zajęć omówiliśmy tylko pełen katalog wzorców projektowych prezentowanych w książce GoF. Celowo nie omawialiśmy w niej przykładów na poziomie kodu źródłowego z następujących powodów:

- aby nie tracić na ogólności zagadnienia
- aby oswoić się z notacją UML
- ze względu na brak wystarczającej wiedzy z Java szczególnie na początku zajęć
- aby uniknąć rozbijania spójności wykładu przykładami gdy wiedza z Java była już wystarczająca

Teraz przyszła kolej na przykłady.

Poniższe omówienie przykładów odwołuje się do przykładów znalezionych na stronie: http://www.fluffycat.com/java-design-patterns

Zostały one dobrane tak, aby odwoływały się do dziedzin życia codziennego i dzięki aby temu były możliwie jak najbardziej intuicyjne.

Nie udało się na razie ani znaleźć ani opracować wersji wszystkich wzorców projektowych GoF bazującej na jednej dziedzinie zastosowań (musiałaby ona być bardzo szeroka), co znacznie uprościłoby proces ich przyswajania przez czytelnika oraz skróciłoby samą prezentację. Opracowanie takiego zbioru przykładów, to sprawa przyszłości.

Omówienie przykładów podzielono na trzy grupy wg klasyfikacji wzorców:

1. Wzorce kreacyjne
    a. restauracja
2. Wzorce strukturalne
    a. przyrządzanie herbaty
    b. robienie wody sodowej o różnych wielkościach i smakach - do zamiany na herbatę o różnych zapachach (już jest) i o różnych smakach (osiąganych przez dolanie soku jak w przykładzie z wodą sodową) - może da się wyeliminować różne rozmiary herbat jeśli podejście to nie naruszy pozostałych klas związanych z przyrządzaniem herbaty
3. Wzorce czynnościowe
    a. produkowanie i katalogowanie płytek DVD

We wszystkich przykładach podano pełny, działający kod źródłowy, dzięki czemu widoczny jest także klient wzorca projektowego niezbyt starannie prezentowany w książce GoF. Warningi mają się pojawiać.

Wszystkie klasy występujace w przykładzie podzielono na następujące grupy:

- klasy dziedziny zastosowań (tzw. domain model)
- klasy wzorca
- klasy klienta (są nimi klasy główne)

W czasie przyszłej optymalizacji przykładów należy dążyć do ograniczenia ilości klas dziedziny (w tym również np. rodzajów zup, smaków, zapachów,...), aby przykład stał się możliwie jak najbardziej spójny i prosty.

Przykłady wymagają Java SE5 lub nowszej.

Wzorce kreacyjne

Bazują one na następującej dziedzinie wiedzy i systemie pojęć:

(...diagram)

1. Singleton

Przeznaczenie (cytat z książki GoF):
Gwarantuje, że klasa ma tylko jeden egzemplarz i zapewnia globalny dostęp do niego.
Przeznaczenie (cytat z fluffycat):
Klasa udostępnia jedyną instancję samej siebie.

Opis: Przykład ilustruje współdzielenie jednej łyżki. Łyżka jest tylko jedna i w danej chwili może używać jej tylko jedna osoba. Oznacza to jedzenie zupy przez więcej niż jedną osobę za pomocą jednej łyżki. Sytuacja nie do pomyślenia w dobrej restauracji! Może to też być np. łyżeczka dedykowana do wsypywania cukru do herbaty umieszczona w cukiernicy. A to już jest powszechnie akceptowane, gdyż się nią nie je!

Problem: Jak zapewnić unikalność egzemplarza łyżeczki. Jak zapewnić mechanizm udostępniania jej innym?

Rozwiązanie: Przez wprowadzenie pola statycznego i odpowiednie zarządzanie jego wartością. Wprowadzenie innej metody zamiast konstruktora klasy reprezentującej łyżkę.

Modyfikacje:
- można korzystając z tego samego mechanizmu zapewnić istnienie jednego pojemnika na łyżki zamiast tylko jendej łyżki - wtedy pole theSpoon powinno być kolekcją a pole theSpoonIsAvaialable powinno być licznikiem
- uwzględnienie wielowątkowości przez uczynienie metody getTheSpoon() metodą synchronizowaną
- dodanie zabezpieczenia przed skopiowaniem łyżki metodą clone() - wystrczy zdefiniować własną metodę clone() zgłaszającą wyjątek lub oddających referencję do istniejącego już obiektu bez tworzenia kopii.

Klasa główna:

_DP_C_GoF_Singleton

```
package pk.dydakt.to.dp.c.gof.singleton;

public class _DP_C_GoF_Singleton {

   public static void main(String[] args) {
       System.out.println("First person getting the spoon");
       SingleSpoon spoonForFirstPerson = SingleSpoon.getTheSpoon();
       if (spoonForFirstPerson != null)
           System.out.println(spoonForFirstPerson);
       else
           System.out.println("No spoon was available");

       System.out.println("");

       System.out.println("Second person getting a spoon");
       SingleSpoon spoonForSecondPerson = SingleSpoon.getTheSpoon();
       if (spoonForSecondPerson != null)
           System.out.println(spoonForSecondPerson);
       else
           System.out.println("No spoon was available");

       System.out.println("");
```

```
        System.out.println("First person returning the spoon");
        SingleSpoon.returnTheSpoon();
        spoonForFirstPerson = null;
        System.out.println("The spoon was returned");

        System.out.println("");

        System.out.println("Second person getting a spoon");
        spoonForSecondPerson = SingleSpoon.getTheSpoon();
        if (spoonForSecondPerson != null)
            System.out.println(spoonForSecondPerson);
        else
            System.out.println("No spoon was available");
    }
}
```

Klasa wzorca:

SingleSpoon

```
package pk.dydakt.to.dp.c.gof.singleton;

import pk.dydakt.to.dp.c.gof.soups.*;

public class SingleSpoon
{
    private Soup soupLastUsedWith;

    private static SingleSpoon theSpoon;
    private static boolean theSpoonIsAvailable = true;

    private SingleSpoon() {}

    public static SingleSpoon getTheSpoon() {
        if (theSpoonIsAvailable) {
            if (theSpoon == null) {
                theSpoon = new SingleSpoon();
            }
            theSpoonIsAvailable = false;
            return theSpoon;
        } else {
            return null;
            //spoon not available,
            //  could throw error or return null (as shown)
        }
    }

    public String toString() {
        return "Behold the glorious single spoon!";
    }

    public static void returnTheSpoon() {
        theSpoon.cleanSpoon();
        theSpoonIsAvailable = true;
    }

    public Soup getSoupLastUsedWith() {
        return this.soupLastUsedWith;
    }
    public void setSoupLastUsedWith(Soup soup) {
        this.soupLastUsedWith = soup;
```

```
    }

    public void cleanSpoon() {
        this.setSoupLastUsedWith(null);
    }
}
```

Wyjście na konsoli:

```
First person getting the spoon
Behold the glorious single spoon!

Second person getting a spoon
No spoon was available

First person returning the spoon
The spoon was returned

Second person getting a spoon
Behold the glorious single spoon!
```

2. Abstract Factory

Przeznaczenie (cytat z książki GoF):
Wzorzec ten zapewnia interfejs umożliwiający tworzenie rodzin powiązanych ze sobą lub zależnych od siebie obiektów bez specyfikowania ich klas konkretnych.
Przeznaczenie (cytat z fluffycat):
Zbiór metod do tworzenia różnych obiektów.
(Wzorzec zawiera różne metody do tworzenia różnych obiektów na różne sposoby.)

The Abstract Factory can make a number of related objects, with each object created individually.

Opis: Przykład ilustruje zamawianie przez klienta zupy dnia od dostawcy zupy z wybranej wcześniej lokalizacji. Może się to odbywać np. poprzez serwis internetowy do zamawiania posiłków z dostawą do domu. Warto zauważyć, że kalendarz jest wspólny dla wszystkich dostawców i dlatego może być umieszczony w klasie klienta. Można jednak ten kalendarz uczynić specyficznym dla poszczególnych dostawców pozbywając się tym samym wiedzy na temat tego, jakie mamy zupy dnia do dyspozycji każdego dnia z klasy klienta. Wiedza ta może być ukryta u dostawcy. Czy jednak takie rozwiązanie nie byłoby nieco ryzykowne dla klienta? Może klient woli najpierw wiedzieć, jaką zupę może wybrać danego dnia, a potem wybrać swojego ulubionego dostawcę tej zupy. Tak właśnie jest w przykładzie, ale właśnie z tego powodu nie pokazuje on pełni zalet tego wzorca.
Można też ten nie tyle przykład, co problem odnieść do zamawiania posiłku w restauracji poprzez skomponowanie dań z menu (w odróżnieniu od wzorca projektowego Factory Method, który kojarzy się raczej z wyborem konkretnego gotowego już zestawu z menu). Może też warto skorzystać z porównania wzorca Builder - tam szef kuchni tworzy zestaw zup dnia na polecenie kierownika restauracji, który zostawia mu wolną rękę - nie wtrąca się do procesu produkcji, a jedynie na końcu, gdy gotowa jest zarówno lista zup dnia jak i same zupy dostaje informację o tym, że wszystko jest gotowe. Tutaj wtrąca się do procesu, ale jakoś nie widzę tego w kodzie!?

Problem: Jak uniezależnić się w kodzie klienta od potrzeby konstruowania docelowych obiektów (zupa dnia)? W jaki sposób przekazać tę funkcjonalność gdzie indziej (fabryki konkretne), aby kod klienta nie musiał znać wszystkich konkretnych klas docelowych obiektów (wszystkie zupy dnia we wszystkich lokalizacjach)?
Można przy pomocy tego wzorca konstruować obiekty złożone (ale też proste). W odróżnieniu od wzorca projektowego Builder kładzie się tu nacisk na rodziny produktów zamiast na tworzenie obiektów krok po kroku, również w odróżnieniu od tamtego wzorca produkt jest przekazywany natychmiast a nie na końcu.
Wzorzec Abstract Factory może być implementowany za pomocą Factory Methods.

Rozwiązanie: Przez stworzenie fabryk zajmujących się tworzeniem obiektów docelowych, w tym fabryki abstrakcyjnej ustalającej wspólny interfejs wszystkich fabryk konkretnych.

Modyfikacje:

Klasa główna:

_DP_C_GoF_AbstractFactory
`package pk.dydakt.to.dp.c.gof.abstractfactory;`

```
import pk.dydakt.to.dp.c.gof.soups.*;

import java.util.Calendar;


class _DP_C_GoF_AbstractFactory {
   public static Soup MakeSoupOfTheDay(AbstractSoupFactory
                                       concreteSoupFactory)
   {
       Calendar todayCalendar = Calendar.getInstance();
       int dayOfWeek = todayCalendar.get(Calendar.DAY_OF_WEEK);

       //int dayOfWeek = Calendar.TUESDAY;

       if (dayOfWeek == Calendar.MONDAY) {
           return concreteSoupFactory.makeChickenSoup();
       } else if (dayOfWeek == Calendar.TUESDAY) {
           return concreteSoupFactory.makeClamChowder();
       } else if (dayOfWeek == Calendar.WEDNESDAY) {
           return concreteSoupFactory.makeFishChowder();
       } else if (dayOfWeek == Calendar.THURSDAY) {
           return concreteSoupFactory.makeMinnestrone();
       } else if (dayOfWeek == Calendar.TUESDAY) {
           return concreteSoupFactory.makePastaFazul();
       } else if (dayOfWeek == Calendar.WEDNESDAY) {
           return concreteSoupFactory.makeTofuSoup();
       } else {
           return concreteSoupFactory.makeVegetableSoup();
       }
   }

   public static void main(String[] args)
   {
       // utworzenie konkretnej fabryki =
       // = wybór lokalizacji dostawcy (albo dostawcy)
       AbstractSoupFactory concreteSoupFactory =
         new BostonConcreteSoupFactory();

       // wybór zupy dnia od danego dostawcy =
       // zdanie się na dostawcę w kwestii wyboru charakterystycznej
       // dla niego zupy dnia
       Soup soupOfTheDay = MakeSoupOfTheDay(concreteSoupFactory);
       System.out.println("The Soup of the day " +
                          concreteSoupFactory.getFactoryLocation() +
                          " is " +
                          soupOfTheDay.getSoupName());

       concreteSoupFactory = new HonoluluConcreteSoupFactory();

       soupOfTheDay = MakeSoupOfTheDay(concreteSoupFactory);
       System.out.println("The Soup of the day " +
                          concreteSoupFactory.getFactoryLocation() +
                          " is " +
                          soupOfTheDay.getSoupName());
   }
}
```

Klasy wzorca:

AbstractSoupFactory

```
package pk.dydakt.to.dp.c.gof.abstractfactory;

import pk.dydakt.to.dp.c.gof.soups.*;

abstract class AbstractSoupFactory {
    String factoryLocation;
    public String getFactoryLocation() {
        return factoryLocation;
    }

    public ChickenSoup makeChickenSoup() {
        return new ChickenSoup();
    }
    public ClamChowder makeClamChowder() {
        return new ClamChowder();
    }
    public FishChowder makeFishChowder() {
        return new FishChowder();
    }
    public Minnestrone makeMinnestrone() {
        return new Minnestrone();
    }
    public PastaFazul makePastaFazul() {
        return new PastaFazul();
    }
    public TofuSoup makeTofuSoup() {
        return new TofuSoup();
    }
    public VegetableSoup makeVegetableSoup() {
        return new VegetableSoup();
    }
}
```

BostonConcreteSoupFactory

```
package pk.dydakt.to.dp.c.gof.abstractfactory;

import pk.dydakt.to.dp.c.gof.soups.*;

class BostonConcreteSoupFactory extends AbstractSoupFactory {
    public BostonConcreteSoupFactory() {
        factoryLocation = "Boston";
    }
    public ClamChowder makeClamChowder() {
        return new BostonClamChowder();
    }
    public FishChowder makeFishChowder() {
        return new BostonFishChowder();
    }
}
```

HonoluluConcreteSoupFactory

```
package pk.dydakt.to.dp.c.gof.abstractfactory;

import pk.dydakt.to.dp.c.gof.soups.*;

class HonoluluConcreteSoupFactory extends AbstractSoupFactory {
    public HonoluluConcreteSoupFactory() {
        factoryLocation = "Honolulu";
```

```
    }
    public ClamChowder makeClamChowder() {
       return new HonoluluClamChowder();
    }
    public FishChowder makeFishChowder() {
       return new HonoluluFishChowder();
    }
}
```

Wyjście na konsoli:

```
The Soup of the day Boston is ScrodFishChowder
The Soup of the day Honolulu is OpakapakaFishChowder
```

3. Factory Method

Przeznaczenie (cytat z książki GoF):
Wzorzec ten określa interfejs do tworzenia obiektów, lecz umożliwia podklasom decydowanie o tym, której klasy ma to być obiekt. Dzięki Factory Method klasy mogą zdać się na podklasy w kwestii tworzenia egzemplarzy.
Przeznaczenie (cytat z fluffycat):
Metody do tworzenia i uzyskiwania elementów jednego obiektu na wiele sposobów.

The Factory Method can make a set of objects, with the objects created only as a set.

Opis: Przykład pokazuje, w jaki sposób klient może zamówić (zobaczyć) zestawy zup oferowane przez poszczególnych dostawców. W tym celu najpierw wybiera dostawcę, a następnie z oferowanych dań wybiera zupy i z zup taką, jakiej sobie życzy.
Innym przykładem może być znalezienie przez klienta w menu zestawu gotowych dań i wybranie takiego zestawu o zadanym numerze, np. „zestaw nr 2 proszę!".
Przykład ten zawiera konkretną fabrykę zdolną do produkcji jednego zbioru obiektów, zatem aby uzyskać taki zestaw obiektów, jaki się chce trzeba utworzyć odpowiednią dla pożądanego zestawu fabrykę.

Problem: Jak uniezależnić kod klienta od informacji zarówno o wszystkich możliwych zestawach u wszystkich możliwych klientów jak i od sposobu konstruowania konkretnych obiektów wchodzących w skład tych zestawów?

Rozwiązanie: Przez stworzenie fabryki tworzącej takie zestawy dań.

Modyfikacje:
- zamiast definiowania konkretnej fabryki zdolnej do produkcji jednego zbioru obiektów można utworzyć fabrykę zdolną do stworzenia zestawu jednego lub więcej dowolnych obiektów określonych przez parametr przekazywany do tej fabryki (por. PHP Design Patterns/Factory Method).

Klasa główna:

_DP_C_GoF_FactoryMethod

```
package pk.dydakt.to.dp.c.gof.factorymethod;

import pk.dydakt.to.dp.c.gof.soups.SoupBuffet;

public class _DP_C_GoF_FactoryMethod {

   public static void main(String[] args) {

      SoupFactoryMethod soupFactoryMethod =
        new SoupFactoryMethod();
      SoupBuffet soupBuffet =
        soupFactoryMethod.makeSoupBuffet();
      soupBuffet.setSoupBuffetName(
        soupFactoryMethod.makeBuffetName());

      soupBuffet.setChickenSoup(
        soupFactoryMethod.makeChickenSoup());
      soupBuffet.setClamChowder(
        soupFactoryMethod.makeClamChowder());
```

```
        soupBuffet.setFishChowder(
          soupFactoryMethod.makeFishChowder());
        soupBuffet.setMinnestrone(
          soupFactoryMethod.makeMinnestrone());
        soupBuffet.setPastaFazul(
          soupFactoryMethod.makePastaFazul());
        soupBuffet.setTofuSoup(
          soupFactoryMethod.makeTofuSoup());
        soupBuffet.setVegetableSoup(
           soupFactoryMethod.makeVegetableSoup());
        System.out.println("At the  " +
                           soupBuffet.getSoupBuffetName() +
                           soupBuffet.getTodaysSoups());

        SoupFactoryMethod bostonSoupFactoryMethod =
          new BostonSoupFactoryMethodSubclass();
        SoupBuffet bostonSoupBuffet =
          bostonSoupFactoryMethod.makeSoupBuffet();
        bostonSoupBuffet.setSoupBuffetName(
          bostonSoupFactoryMethod.makeBuffetName());

        bostonSoupBuffet.setChickenSoup(
          bostonSoupFactoryMethod.makeChickenSoup());
        bostonSoupBuffet.setClamChowder(
          bostonSoupFactoryMethod.makeClamChowder());
        bostonSoupBuffet.setFishChowder(
          bostonSoupFactoryMethod.makeFishChowder());
        bostonSoupBuffet.setMinnestrone(
          bostonSoupFactoryMethod.makeMinnestrone());
        bostonSoupBuffet.setPastaFazul(
          bostonSoupFactoryMethod.makePastaFazul());
        bostonSoupBuffet.setTofuSoup(
          bostonSoupFactoryMethod.makeTofuSoup());
        bostonSoupBuffet.setVegetableSoup(
          bostonSoupFactoryMethod.makeVegetableSoup());
        System.out.println("At the  " +
                           bostonSoupBuffet.getSoupBuffetName() +
                           bostonSoupBuffet.getTodaysSoups());

        SoupFactoryMethod honoluluSoupFactoryMethod =
          new HonoluluSoupFactoryMethodSubclass();
        SoupBuffet honoluluSoupBuffet =
          honoluluSoupFactoryMethod.makeSoupBuffet();
        honoluluSoupBuffet.setSoupBuffetName(
          honoluluSoupFactoryMethod.makeBuffetName());

        honoluluSoupBuffet.setChickenSoup(
          honoluluSoupFactoryMethod.makeChickenSoup());
        honoluluSoupBuffet.setClamChowder(
          honoluluSoupFactoryMethod.makeClamChowder());
        honoluluSoupBuffet.setFishChowder(
          honoluluSoupFactoryMethod.makeFishChowder());
        honoluluSoupBuffet.setMinnestrone(
          honoluluSoupFactoryMethod.makeMinnestrone());
        honoluluSoupBuffet.setPastaFazul(
          honoluluSoupFactoryMethod.makePastaFazul());
        honoluluSoupBuffet.setTofuSoup(
          honoluluSoupFactoryMethod.makeTofuSoup());
        honoluluSoupBuffet.setVegetableSoup(
          honoluluSoupFactoryMethod.makeVegetableSoup());
        System.out.println("At the  " +
```

```
                            honoluluSoupBuffet.getSoupBuffetName() +
                            honoluluSoupBuffet.getTodaysSoups());
    }
}
```

Klasy wzorca:

SoupFactoryMethod

```
package pk.dydakt.to.dp.c.gof.factorymethod;

import pk.dydakt.to.dp.c.gof.soups.*;

class SoupFactoryMethod {
    public SoupFactoryMethod() {}

    public SoupBuffet makeSoupBuffet() {
        return new SoupBuffet();
    }

    public ChickenSoup makeChickenSoup() {
        return new ChickenSoup();
    }
    public ClamChowder makeClamChowder() {
        return new ClamChowder();
    }
    public FishChowder makeFishChowder() {
        return new FishChowder();
    }
    public Minnestrone makeMinnestrone() {
        return new Minnestrone();
    }
    public PastaFazul makePastaFazul() {
        return new PastaFazul();
    }
    public TofuSoup makeTofuSoup() {
        return new TofuSoup();
    }
    public VegetableSoup makeVegetableSoup() {
        return new VegetableSoup();
    }

    public String makeBuffetName() {
        return "Soup Buffet";
    }
}
```

BostonSoupFactoryMethodSubclass

```
package pk.dydakt.to.dp.c.gof.factorymethod;

import pk.dydakt.to.dp.c.gof.soups.*;

class BostonSoupFactoryMethodSubclass extends SoupFactoryMethod {
    public String makeBuffetName() {
        return "Boston Soup Buffet";
    }
    public ClamChowder makeClamChowder() {
        return new BostonClamChowder();
```

```
    }
    public FishChowder makeFishChowder() {
        return new BostonFishChowder();
    }
}
```

HonoluluSoupFactoryMethodSubclass

```
package pk.dydakt.to.dp.c.gof.factorymethod;

import pk.dydakt.to.dp.c.gof.soups.*;

class HonoluluSoupFactoryMethodSubclass extends SoupFactoryMethod {
    public String makeBuffetName() {
        return "Honolulu Soup Buffet";
    }
    public ClamChowder makeClamChowder() {
        return new HonoluluClamChowder();
    }
    public FishChowder makeFishChowder() {
        return new HonoluluFishChowder();
    }
}
```

Wyjście na konsoli:

```
At the Soup Buffet Today's Soups!
Chicken Soup: ChickenSoup
Clam Chowder: ClamChowder
Fish Chowder: FishChowder
Minnestrone: Minestrone
Pasta Fazul: Pasta Fazul
Tofu Soup: Tofu Soup
Vegetable Soup: Vegetable Soup

At the Boston Soup Buffet Today's Soups!
Chicken Soup: ChickenSoup
Clam Chowder: QuahogChowder
Fish Chowder: ScrodFishChowder
Minnestrone: Minestrone
Pasta Fazul: Pasta Fazul
Tofu Soup: Tofu Soup
Vegetable Soup: Vegetable Soup

At the Honolulu Soup Buffet Today's Soups!
Chicken Soup: ChickenSoup
Clam Chowder: PacificClamChowder
Fish Chowder: OpakapakaFishChowder
Minnestrone: Minestrone
Pasta Fazul: Pasta Fazul
Tofu Soup: Tofu Soup
Vegetable Soup: Vegetable Soup
```

4. Builder

Przeznaczenie (cytat z książki GoF):
Wzorzec ten oddziela konstrukcję obiektów złożonych od ich reprezentacji, umożliwiając tym samym powstanie w jednym procesie konstrukcyjnym różnych reprezentacji.
Przeznaczenie (cytat z fluffycat):
Tworzy i udostępnia jeden obiekt na wiele sposobów.

Opis: Przykład pokazuje jak stworzyć obiekt złożony, jakim jest zestaw zup dnia. Obiekt ten jest w odróżnieniu od wzorca projektowego Abstract Factory tworzony krok po kroku i zwracany dopiero po utworzeniu całości. Odpowiada to sytuacji, w której np. kierownik danej lokalizacji sieci restauracji wydaje szefowi kuchni polecenie opracowania i stworzenia na dany dzień zestawu zup dnia. Dla każdej grupy zup trzeba określić jej reprezentanta, który będzie zupą dnia. Szef kuchni ma pełną swobodę zarówno w określaniu zup dnia jak i w ich przyrządzeniu. Gdy lista zup dnia jest gotowa szef kuchni zleca ich przygotowanie, a następnie, gdy zupy są już gotowe informuje kierownika o tym fakcie udostępniając mu również listę zup. W ten sposób nie angażuje kierownika we własny proces decyzyjny - kierownik oczekuje wykonania zadania w całości.

Problem: Jak budowniczy może stworzyć złożone produkty i udostępnić je kierownikowi w całości, gdy już będą gotowe.

Rozwiązanie:

Klasa główna:

_DP_C_GoF_Builder

```
package pk.dydakt.to.dp.c.gof.builder;

import pk.dydakt.to.dp.c.gof.soups.*;

public class _DP_C_GoF_Builder {

   public static SoupBuffet CreateSoupBuffet(
     SoupBuffetBuilder soupBuffetBuilder) {
        soupBuffetBuilder.buildSoupBuffet();

        soupBuffetBuilder.setSoupBuffetName();

        soupBuffetBuilder.buildChickenSoup();
        soupBuffetBuilder.buildClamChowder();
        soupBuffetBuilder.buildFishChowder();
        soupBuffetBuilder.buildMinnestrone();
        soupBuffetBuilder.buildPastaFazul();
        soupBuffetBuilder.buildTofuSoup();
        soupBuffetBuilder.buildVegetableSoup();

        return soupBuffetBuilder.getSoupBuffet();
   }

   public static void main(String[] args) {

       SoupBuffet bostonSoupBuffet =
         CreateSoupBuffet(new BostonSoupBuffetBuilder());
       System.out.println("At the " +
                          bostonSoupBuffet.getSoupBuffetName() +
```

```
                            bostonSoupBuffet.getTodaysSoups());

        SoupBuffet honoluluSoupBuffet =
          CreateSoupBuffet(new HonoluluSoupBuffetBuilder());
        System.out.println("At the " +
                            honoluluSoupBuffet.getSoupBuffetName() +
                            honoluluSoupBuffet.getTodaysSoups());
    }
}
```

Klasy wzorca:

SoupBuffetBuilder

```
package pk.dydakt.to.dp.c.gof.builder;

import pk.dydakt.to.dp.c.gof.soups.*;

abstract class SoupBuffetBuilder {
    SoupBuffet soupBuffet;

    public SoupBuffet getSoupBuffet() {
        return soupBuffet;
    }

    public void buildSoupBuffet() {
        soupBuffet = new SoupBuffet();
    }

    public abstract void setSoupBuffetName();

    public void buildChickenSoup() {
        soupBuffet.chickenSoup = new ChickenSoup();
    }
    public void buildClamChowder() {
        soupBuffet.clamChowder = new ClamChowder();
    }
    public void buildFishChowder() {
        soupBuffet.fishChowder = new FishChowder();
    }
    public void buildMinnestrone() {
        soupBuffet.minnestrone = new Minnestrone();
    }
    public void buildPastaFazul() {
        soupBuffet.pastaFazul = new PastaFazul();
    }
    public void buildTofuSoup() {
        soupBuffet.tofuSoup = new TofuSoup();
    }
    public void buildVegetableSoup() {
        soupBuffet.vegetableSoup = new VegetableSoup();
    }
}
```

BostonSoupBuffetBuilder

```
package pk.dydakt.to.dp.c.gof.builder;

import pk.dydakt.to.dp.c.gof.soups.BostonClamChowder;
import pk.dydakt.to.dp.c.gof.soups.BostonFishChowder;

class BostonSoupBuffetBuilder extends SoupBuffetBuilder {
```

```
    public void buildClamChowder() {
       soupBuffet.clamChowder = new BostonClamChowder();
    }
    public void buildFishChowder() {
       soupBuffet.fishChowder = new BostonFishChowder();
    }

    public void setSoupBuffetName() {
       soupBuffet.soupBuffetName = "Boston Soup Buffet";
    }
}
```

HonoluluSoupBuffetBuilder

```
package pk.dydakt.to.dp.c.gof.builder;

import pk.dydakt.to.dp.c.gof.soups.HonoluluClamChowder;
import pk.dydakt.to.dp.c.gof.soups.HonoluluFishChowder;

class HonoluluSoupBuffetBuilder extends SoupBuffetBuilder {
    public void buildClamChowder() {
        soupBuffet.clamChowder = new HonoluluClamChowder();
    }
    public void buildFishChowder() {
        soupBuffet.fishChowder = new HonoluluFishChowder();
    }

    public void setSoupBuffetName() {
        soupBuffet.soupBuffetName = "Honolulu Soup Buffet";
    }
}
```

Wyjście na konsoli:

```
At the Boston Soup Buffet Today's Soups!
Chicken Soup: ChickenSoup
Clam Chowder: QuahogChowder
Fish Chowder: Scrod FishChowder
Minnestrone: Minestrone
Pasta Fazul: Pasta Fazul
Tofu Soup: Tofu Soup
Vegetable Soup: Vegetable Soup

At the Honolulu Soup Buffet Today's Soups!
Chicken Soup: ChickenSoup
Clam Chowder: PacificClamChowder
Fish Chowder: OpakapakaFishChowder
Minnestrone: Minestrone
Pasta Fazul: Pasta Fazul
Tofu Soup: Tofu Soup
Vegetable Soup: Vegetable Soup
```

5. Prototype

Przeznaczenie (cytat z książki GoF):
Wzorzec ten specyfikuje rodzaje tworzonych obiektów, używając prototypowego egzemplarza, a także tworzy nowe obiekty, kopiując ten prototyp.
Przeznaczenie (cytat z fluffycat):
Tworzy nowe obiekty przez sklonowanie obiektów ustanowionych jako prototypy.

Opis: Przykład może wyglądać następująco. Jest urządzane przyjęcie z przygotowaną wcześniej listą dań. Szef kelnerów wie zatem jakie sztućce należy podać do zestawu wszystkim gościom - ma odpowiednie kwalifikacje. Nie wiedzą tego jednak nowo przyjęci kelnerzy, na dodatek nie mają zbyt dobrej pamięci. Dlatego szef kelnerów musi pokazać im, jakie sztućce należy rozłożyć na stole. Nie może podać więcej niż dwóch sztućców na raz, bo kelnerzy nie zapamiętają albo się pomylą. Tak więc gdy kelner zobaczy sztućce idzie do pojemnika z nimi (PrototypeFactory) wyszukując pojemnik z dokładnie takimi sztućcami jakie pokazał mu szef i wybiera odpowiednią ilość. Następnie zanosi je na stół.

Problem:

Rozwiązanie:

Klasa główna:

_DP_C_GoF_Prototype

```
package pk.dydakt.to.dp.c.gof.prototype;

public class _DP_C_GoF_Prototype {

   public static void main(String[] args) {
       System.out.println(
         "Creating a Prototype Factory with " +
         " a SoupSpoon and a SaladFork");
       PrototypeFactory prototypeFactory =
         new PrototypeFactory(new SoupSpoon(), new SaladFork());
       AbstractSpoon spoon =
         prototypeFactory.makeSpoon();
       AbstractFork fork =
         prototypeFactory.makeFork();
       System.out.println("Getting the Spoon and Fork name:");
       System.out.println("Spoon: " + spoon.getSpoonName() +
                          ", Fork: " + fork.getForkName());
       System.out.println(" ");
       System.out.println("Creating a Prototype Factory " +
                          "with a SaladSpoon and a SaladFork");
       prototypeFactory =
         new PrototypeFactory(new SaladSpoon(), new SaladFork());
       spoon = prototypeFactory.makeSpoon();
       fork = prototypeFactory.makeFork();
       System.out.println("Getting the Spoon and Fork name:");
       System.out.println("Spoon: " + spoon.getSpoonName() +
                          ", Fork: " + fork.getForkName());
    }
}
```

Klasy wzorca:

PrototypeFactory

```
package pk.dydakt.to.dp.c.gof.prototype;
```

```
public class PrototypeFactory {
    AbstractSpoon prototypeSpoon;
    AbstractFork prototypeFork;

    public PrototypeFactory(AbstractSpoon spoon, AbstractFork fork) {
       prototypeSpoon = spoon;
       prototypeFork = fork;
   }

   public AbstractSpoon makeSpoon() {
       return (AbstractSpoon)prototypeSpoon.clone();
   }
   public AbstractFork makeFork() {
       return (AbstractFork)prototypeFork.clone();
   }
}
```

AbstractFork

```
package pk.dydakt.to.dp.c.gof.prototype;

public abstract class AbstractFork implements Cloneable
{
   String forkName;

   public void setForkName(String forkName) {
       this.forkName = forkName;
   }
   public String getForkName() {
       return this.forkName;
   }

   public Object clone()
   {
       Object object = null;
       try {
           object = super.clone();
       } catch (CloneNotSupportedException exception) {
           System.err.println("AbstractFork is not Cloneable");
       }
       return object;
   }
}
```

AbstractSpoon

```
package pk.dydakt.to.dp.c.gof.prototype;

public abstract class AbstractSpoon implements Cloneable {
   String spoonName;

   public void setSpoonName(String spoonName) {
       this.spoonName = spoonName;
   }
   public String getSpoonName() {
       return this.spoonName;
   }

   public Object clone() {
       Object object = null;
       try {
```

```
            object = super.clone();
        } catch (CloneNotSupportedException exception) {
            System.err.println("AbstractSpoon is not Cloneable");
        }
        return object;
    }
}
```

SaladFork

```
package pk.dydakt.to.dp.c.gof.prototype;

public class SaladFork extends AbstractFork {
    public SaladFork() {
        setForkName("Salad Fork");
    }
}
```

SaladSpoon

```
package pk.dydakt.to.dp.c.gof.prototype;

public class SaladSpoon extends AbstractSpoon {
    public SaladSpoon() {
        setSpoonName("Salad Spoon");
    }
}
```

SoupSpoon

```
package pk.dydakt.to.dp.c.gof.prototype;

public class SoupSpoon extends AbstractSpoon {
    public SoupSpoon() {
        setSpoonName("Soup Spoon");
    }
}
```

Wyjście na konsoli:

```
Creating a Prototype Factory with  a SoupSpoon and a SaladFork
Getting the Spoon and Fork name:
Spoon: Soup Spoon, Fork: Salad Fork

Creating a Prototype Factory with a SaladSpoon and a SaladFork
Getting the Spoon and Fork name:
Spoon: Salad Spoon, Fork: Salad Fork
```

Bazują one na następującej dziedzinie wiedzy i systemie pojęć:

(...diagram)

1. Decorator

Przeznaczenie (cytat z książki GoF):
Dynamicznie dołącza do obiektów dodatkowe zobowiązania. Zapewnia elastyczną alternatywę dla tworzenia podklas w celu rozszerzenia funkcjonalności.
Przeznaczenie (cytat z fluffycat):
Jedna klasa przejmuje inną klasę, z których obie rozszerzają tę samą klasę abstrakcyjną oraz dodaje funcjonalność.

Opis: Przykład może być następujący. Klient zamawia herbatę o jakiejś pięknie brzmiącej nazwie. Parzeniem herbat w restauracji zajmuje się specjalna osoba (ChaiDecorator). Dysponuje ona zarówno listą wszystkich możliwych dodatków do herbaty jak i listą składników (wraz z ilością), jakie należy dodać do każdego z rodzajów herbaty. Gdy przygotuje wszystkie stosowne składniki, wtedy zaparza zamówioną herbatę dla klienta.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_S_GoF_Decorator

```
package pk.dydakt.to.dp.s.gof.decorator;

public class _DP_S_GoF_Decorator {

   public static void main(String[] args) {
       Tea teaLeaves = new TeaLeaves();
       Tea chaiDecorator = new ChaiDecorator(teaLeaves);
       chaiDecorator.steepTea();
   }
}
```

Klasy wzorca:

Tea

```
package pk.dydakt.to.dp.s.gof.decorator;

public abstract class Tea {
   boolean teaIsSteeped;

   public abstract void steepTea();
}
```

TeaLeaves

```
package pk.dydakt.to.dp.s.gof.decorator;

public class TeaLeaves extends Tea {
```

```
    public TeaLeaves() {
        teaIsSteeped = false;
    }

    public void steepTea() {
        teaIsSteeped = true;
        System.out.println("tea leaves are steeping");
    }
}
```

ChaiDecorator

```
package pk.dydakt.to.dp.s.gof.decorator;

import java.util.ArrayList;
import java.util.ListIterator;

public class ChaiDecorator extends Tea {
    private Tea teaToMakeChai;
    private ArrayList<String> chaiIngredients = new ArrayList<String>();

    public ChaiDecorator(Tea teaToMakeChai) {
        this.addTea(teaToMakeChai);
        chaiIngredients.add("bay leaf");
        chaiIngredients.add("cinnamon stick");
        chaiIngredients.add("ginger");
        chaiIngredients.add("honey");
        chaiIngredients.add("soy milk");
        chaiIngredients.add("vanilla bean");
    }

    private void addTea(Tea teaToMakeChaiIn) {
        this.teaToMakeChai = teaToMakeChaiIn;
    }

    public void steepTea() {
        this.steepChai();
    }

    public void steepChai() {
        teaToMakeChai.steepTea();
        this.steepChaiIngredients();
        System.out.println("tea is steeping with chai");
    }

    public void steepChaiIngredients() {
        ListIterator listIterator = chaiIngredients.listIterator();
        while (listIterator.hasNext()) {
            System.out.println(((String)(listIterator.next())) +
                                        " is steeping");
        }
        System.out.println("chai ingredients are steeping");
    }
}
```

Wyjście na konsoli:

```
tea leaves are steeping
bay leaf is steeping
cinnamon stick is steeping
ginger is steeping
honey is steeping
soy milk is steeping
vanilla bean is steeping
```

```
chai ingredients are steeping
tea is steeping with chai
```

2. Composite

Przeznaczenie (cytat z książki GoF):
Składa obiekty w struktury drzewiaste reprezentujące hierarchie typu część-całość. Umożliwia klientom jednakowe traktowanie pojedynczych obiektów i złożeń obiektów.
Przeznaczenie (cytat z fluffycat):
Łączy grupy obiketów o tej samej sygnaturze.

Opis: Przykład ilustruje pakowanie torebek herbaty do pudełek. Może ono mieć miejsce w restauracji, gdy herbata jest przepakowywana do pudełek firmowych, które np. są dostarczane wraz z zawartością do klienta. Pudełka są różnych rozmiarów, zatem duże pudełka mogą w sobie zawierać zarówno torebki herbaty jak również małe pudełka z torebkami herbaty. Małe pudełka przydają się do oddzielenia od siebie herbat jednego rodzaju w celu nieprzekazywania zapachów pomiedzy nimi.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_S_GoF_Composite

```
package pk.dydakt.to.dp.s.gof.composite;

public class _DP_S_GoF_Composite {

   public static void main(String[] args) {
       System.out.println("Creating tinOfTeaBags");
       TeaBags tinOfTeaBags =
           new TinOfTeaBags("tin of tea bags");
       System.out.println("The tinOfTeaBags has " +
                          tinOfTeaBags.countTeaBags() +
                          " tea bags in it.");

       System.out.println(" ");

       System.out.println("Creating teaBag1");
       TeaBags teaBag1 = new OneTeaBag("tea bag 1");
       System.out.println("The teaBag1 has " +
                          teaBag1.countTeaBags() +
                          " tea bags in it.");

       System.out.println(" ");

       System.out.println("Creating teaBag2");
       TeaBags teaBag2 = new OneTeaBag("tea bag 2");
       System.out.println("The teaBag2 has " +
                          teaBag2.countTeaBags() +
                          " tea bags in it.");

       System.out.println(" ");

       System.out.println(
         "Putting teaBag1 and teaBag2 in tinOfTeaBags");
       if (tinOfTeaBags.add(teaBag1)) {
          System.out.println(
            "teaBag1 added successfully to tinOfTeaBags");
       } else {
```

```
            System.out.println(
               "teaBag1 not added successfully tinOfTeaBags");
         }
         if (tinOfTeaBags.add(teaBag2)) {
            System.out.println(
              "teaBag2 added successfully to tinOfTeaBags");
         } else {
            System.out.println(
              "teaBag2 not added successfully tinOfTeaBags");
         }
         System.out.println("The tinOfTeaBags now has " +
                            tinOfTeaBags.countTeaBags() +
                            " tea bags in it.");

         System.out.println(" ");

         System.out.println("Creating smallTinOfTeaBags");
         TeaBags smallTinOfTeaBags =
           new TinOfTeaBags("small tin of tea bags");
         System.out.println("The smallTinOfTeaBags has " +
                            smallTinOfTeaBags.countTeaBags() +
                            " tea bags in it.");
         System.out.println("Creating teaBag3");
         TeaBags teaBag3 =
           new OneTeaBag("tea bag 3");
         System.out.println("The teaBag3 has " +
                            teaBag3.countTeaBags() +
                            " tea bags in it.");
         System.out.println("Putting teaBag3 in smallTinOfTeaBags");
         if (smallTinOfTeaBags.add(teaBag3)) {
             System.out.println(
               "teaBag3 added successfully to smallTinOfTeaBags");
         } else {
             System.out.println(
               "teaBag3 not added successfully to smallTinOfTeaBags");
         }
         System.out.println("The smallTinOfTeaBags now has " +
                            smallTinOfTeaBags.countTeaBags() +
                            " tea bags in it.");

         System.out.println(" ");

         System.out.println(
           "Putting smallTinOfTeaBags in tinOfTeaBags");
         if (tinOfTeaBags.add(smallTinOfTeaBags)) {
             System.out.println(
               "smallTinOfTeaBags added successfully to tinOfTeaBags");
         } else {
             System.out.println(
               "smallTinOfTeaBags not added successfully to tinOfTeaBags");
         }
         System.out.println("The tinOfTeaBags now has " +
                            tinOfTeaBags.countTeaBags() +
                            " tea bags in it.");

         System.out.println(" ");

         System.out.println("Removing teaBag2 from tinOfTeaBags");
         if (tinOfTeaBags.remove(teaBag2)) {
             System.out.println(
               "teaBag2 successfully removed from tinOfTeaBags");
```

```
        } else {
            System.out.println(
              "teaBag2 not successfully removed from tinOfTeaBags");
        }
        System.out.println("The tinOfTeaBags now has " +
                           tinOfTeaBags.countTeaBags() +
                           " tea bags in it.");
    }
}
```

Klasy wzorca:

TeaBags

```
package pk.dydakt.to.dp.s.gof.composite;

import java.util.LinkedList;
import java.util.ListIterator;

public abstract class TeaBags {
    LinkedList<TeaBags> teaBagList;
    TeaBags parent;
    String name;

    public abstract int countTeaBags();

    public abstract boolean add(TeaBags teaBagsToAdd);
    public abstract boolean remove(TeaBags teaBagsToRemove);
    public abstract ListIterator createListIterator();

    public void setParent(TeaBags parentIn) {
        parent = parentIn;
    }
    public TeaBags getParent() {
       return parent;
    }

    public void setName(String nameIn) {
        name = nameIn;
    }
    public String getName() {
        return name;
    }
}
```

OneTeaBag

```
package pk.dydakt.to.dp.s.gof.composite;

import java.util.ListIterator;

public class OneTeaBag extends TeaBags {
    public OneTeaBag(String nameIn) {
        this.setName(nameIn);
    }

    public int countTeaBags() {
        return 1;
    }

    public boolean add(TeaBags teaBagsToAdd) {
        return false;
    }
```

```
    public boolean remove(TeaBags teaBagsToRemove) {
        return false;
    }
    public ListIterator createListIterator() {
        return null;
    }
}
```

TinOfTeaBags

```
package pk.dydakt.to.dp.s.gof.composite;

import java.util.LinkedList;
import java.util.ListIterator;

public class TinOfTeaBags extends TeaBags {
   public TinOfTeaBags(String nameIn) {
       teaBagList = new LinkedList<TeaBags>();
       this.setName(nameIn);
   }

   public int countTeaBags() {
       int totalTeaBags = 0;
       ListIterator listIterator = this.createListIterator();
       TeaBags tempTeaBags;
       while (listIterator.hasNext()) {
           tempTeaBags = (TeaBags)listIterator.next();
           totalTeaBags += tempTeaBags.countTeaBags();
       }
       return totalTeaBags;
   }

   public boolean add(TeaBags teaBagsToAdd) {
       teaBagsToAdd.setParent(this);
       return teaBagList.add(teaBagsToAdd);
   }

   public boolean remove(TeaBags teaBagsToRemove) {
       ListIterator listIterator =
           this.createListIterator();
       TeaBags tempTeaBags;
       while (listIterator.hasNext()) {
           tempTeaBags = (TeaBags)listIterator.next();
           if (tempTeaBags == teaBagsToRemove) {
               listIterator.remove();
               return true;
           }
       }
       return false;
   }

   public ListIterator createListIterator() {
       ListIterator listIterator = teaBagList.listIterator();
       return listIterator;
   }
}
```

Wyjście na konsoli:

```
Creating tinOfTeaBags
The tinOfTeaBags has 0 tea bags in it.

Creating teaBag1
```

```
The teaBag1 has 1 tea bags in it.

Creating teaBag2
The teaBag2 has 1 tea bags in it.

Putting teaBag1 and teaBag2 in tinOfTeaBags
teaBag1 added successfully to tinOfTeaBags
teaBag2 added successfully to tinOfTeaBags
The tinOfTeaBags now has 2 tea bags in it.

Creating smallTinOfTeaBags
The smallTinOfTeaBags has 0 tea bags in it.
Creating teaBag3
The teaBag3 has 1 tea bags in it.
Putting teaBag3 in smallTinOfTeaBags
teaBag3 added successfully to smallTinOfTeaBags
The smallTinOfTeaBags now has 1 tea bags in it.

Putting smallTinOfTeaBags in tinOfTeaBags
smallTinOfTeaBags added successfully to tinOfTeaBags
The tinOfTeaBags now has 3 tea bags in it.

Removing teaBag2 from tinOfTeaBags
teaBag2 successfully removed from tinOfTeaBags
The tinOfTeaBags now has 2 tea bags in it.
```

3. Adapter

Przeznaczenie (cytat z książki GoF):
Przekształca interfejs klasy w taki, jakiego oczekują klienci. Dzięki wzorcowi klasy mogą współpracować, co bez niego nie byłoby możliwe, ponieważ mają niezgodne interfejsy.
Przeznaczenie (cytat z fluffycat):
Klasa rozszerza inną klasę, przejmuje obiekt i sprawia, że zachowuje się on jak klasa rozszerzona.

Opis: Przykład ilustruje parzenie herbaty w filiżance. Można to zrobić za pomocą torebki herbaty, która zawiera tyle herbaty ile potrzeba na jedną filiżankę. Jeśli jednak chcielibyśmy zaparzyć herbatę z liści, to możemy to zrobić zarówno w filiżance jak i np. w czajniczku. Zaparzanie herbaty z liści obsługuje się więc inaczej niż zaparzanie z torebki. Aby zaparzyć herbatę z liści w filiżance należy więc odpowiednio dopasować sposób parzenia do faktu parzenia jej w filiżance. Służy do tego kulka do parzenia herbaty z liści dopasowująca parzenie z liści do potrzeb parzenia filiżanki. Dzięki zastosowaniu takiej kulki możemy zarówno załadować odpowiednią dawkę herbaty jak i nie naśmiecić zbytnio liśćmi w filiżance.

Problem:

Rozwiązanie:

Klasa główna:

_DP_S_GoF_Adapter

```
package pk.dydakt.to.dp.s.gof.adapter;

public class _DP_S_GoF_Adapter {

   public static void main(String[] args) {
       TeaCup teaCup = new TeaCup();

       System.out.println("Steeping tea bag");
       TeaBag teaBag = new TeaBag();
       teaCup.steepTeaBag(teaBag);

       System.out.println("Steeping loose leaf tea");
       LooseLeafTea looseLeafTea = new LooseLeafTea();
       TeaBall teaBall = new TeaBall(looseLeafTea);
       teaCup.steepTeaBag(teaBall);
   }
}
```

Klasy wzorca:

TeaCup

```
package pk.dydakt.to.dp.s.gof.adapter;

public class TeaCup {
   public void steepTeaBag(TeaBag teaBag) {
       teaBag.steepTeaInCup();
   }
}
```

TeaBag

```
package pk.dydakt.to.dp.s.gof.adapter;
```

```
public class TeaBag {
   boolean teaBagIsSteeped;

   public TeaBag() {
       teaBagIsSteeped = false;
   }

   public void steepTeaInCup() {
       teaBagIsSteeped = true;
       System.out.println("tea bag is steeping in cup");
   }
}
```

LooseLeafTea

```
package pk.dydakt.to.dp.s.gof.adapter;

public class LooseLeafTea {
   boolean teaIsSteeped;

   public LooseLeafTea() {
       teaIsSteeped = false;
   }

   public void steepTea() {
       teaIsSteeped = true;
       System.out.println("tea is steeping");
   }
}
```

TeaBall

```
package pk.dydakt.to.dp.s.gof.adapter;

public class TeaBall extends TeaBag {
   LooseLeafTea looseLeafTea;

   public TeaBall(LooseLeafTea looseLeafTeaIn) {
       looseLeafTea = looseLeafTeaIn;
       teaBagIsSteeped = looseLeafTea.teaIsSteeped;
   }

   public void steepTeaInCup() {
       looseLeafTea.steepTea();
       teaBagIsSteeped = true;
   }
}
```

Wyjście na konsoli:

```
Steeping tea bag
tea bag is steeping in cup
Steeping loose leaf tea
tea is steeping
```

4. Bridge

Przeznaczenie (cytat z książki GoF):
Oddziela abstrakcję od jej implementacji, tak by mogły się zmieniać niezależnie jedna od drugiej.
Przeznaczenie (cytat z fluffycat):
Abstrakcja i implementacja znajdują się w różnych hierarchiach klas.

Opis: Przykład przyrządzania wody sodowej o różnych smakach. Smak to implementacja, którą wybieramy najpierw. Na wybranej implementacji możemy eksperymentować poprzez abstrakcję, jaką jest wybór ilości wody sodowej. Ilość może być dowolna dzięki odpowiedniej implementacji zapewniającej stężenie soku w wodzie sodowej niezależne od ilości przyrządzonego napoju. Można dodawać nowe smaki (zmiana implementacji) niezależnie od sposobu zarządzania ilością napoju (stała abstrakcja). Można też zmieniać ilość przyrządzanego napoju (zmiana abstrakcji) niezależnie od smaków jakimi dysponujemy (stała implementacja). Dzięki takiej separacji różnych zjawisk otrzymujemy elastyczność w kreowaniu różnych napojów. Nb. możemy się pokusić o zmianę stężenia soku?

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_S_GoF_Bridge

```
package pk.dydakt.to.dp.s.gof.bridge;

public class _DP_S_GoF_Bridge {

   public static void testCherryPlatform() {
       SodaImpSingleton sodaImpSingleton =
         new SodaImpSingleton(new CherrySodaImp());
       System.out.println(
         "testing medium soda on the cherry platform");
       MediumSoda mediumSoda = new MediumSoda();
       mediumSoda.pourSoda();
       System.out.println(
         "testing super size soda on the cherry platform");
       SuperSizeSoda superSizeSoda = new SuperSizeSoda();
       superSizeSoda.pourSoda();
   }

   public static void testGrapePlatform() {
       SodaImpSingleton sodaImpSingleton =
         new SodaImpSingleton(new GrapeSodaImp());
       System.out.println(
         "testing medium soda on the grape platform");
       MediumSoda mediumSoda = new MediumSoda();
       mediumSoda.pourSoda();
       System.out.println(
         "testing super size soda on the grape platform");
       SuperSizeSoda superSizeSoda = new SuperSizeSoda();
       superSizeSoda.pourSoda();
   }

   public static void testOrangePlatform() {
       SodaImpSingleton sodaImpSingleton =
```

```
          new SodaImpSingleton(new OrangeSodaImp());
        System.out.println(
          "testing medium soda on the orange platform");
        MediumSoda mediumSoda = new MediumSoda();
        mediumSoda.pourSoda();
        System.out.println(
          "testing super size soda on the orange platform");
        SuperSizeSoda superSizeSoda = new SuperSizeSoda();
        superSizeSoda.pourSoda();
    }

    public static void main(String[] args) {
         testCherryPlatform();
         testGrapePlatform();
         testOrangePlatform();
    }
}
```

Klasy wzorca:

Soda

```
package pk.dydakt.to.dp.s.gof.bridge;

public abstract class Soda {
    SodaImp sodaImp;

    public void setSodaImp() {
        this.sodaImp = SodaImpSingleton.getTheSodaImp();
    }
    public SodaImp getSodaImp() {
        return this.sodaImp;
    }

    public abstract void pourSoda();
}
```

SodaImp

```
package pk.dydakt.to.dp.s.gof.bridge;

public abstract class SodaImp {
    public abstract void pourSodaImp();
}
```

SodaImpSingleton

```
package pk.dydakt.to.dp.s.gof.bridge;

public class SodaImpSingleton {
    private static SodaImp sodaImp;

    public SodaImpSingleton(SodaImp sodaImpIn) {
        SodaImpSingleton.sodaImp = sodaImpIn;
    }

    public static SodaImp getTheSodaImp() {
        return sodaImp;
    }
}
```

MediumSoda

```
package pk.dydakt.to.dp.s.gof.bridge;
```

```
public class MediumSoda extends Soda {
   public MediumSoda() {
       setSodaImp();
   }

   public void pourSoda() {
       SodaImp sodaImp = this.getSodaImp();
       for (int i = 0; i < 2; i++) {
           System.out.print("...glug...");
           sodaImp.pourSodaImp();
       }
       System.out.println(" ");
   }
}
```

SuperSizeSoda

```
package pk.dydakt.to.dp.s.gof.bridge;

public class SuperSizeSoda extends Soda {
   public SuperSizeSoda() {
       setSodaImp();
   }

   public void pourSoda() {
       SodaImp sodaImp = this.getSodaImp();
       for (int i = 0; i < 5; i++) {
           System.out.print("...glug...");
           sodaImp.pourSodaImp();
       }
       System.out.println(" ");
   }
}
```

CherrySodaImp

```
package pk.dydakt.to.dp.s.gof.bridge;

public class CherrySodaImp extends SodaImp {
   CherrySodaImp() {}

   public void pourSodaImp() {
       System.out.println("Yummy Cherry Soda!");
   }
}
```

GrapeSodaImp

```
package pk.dydakt.to.dp.s.gof.bridge;

public class GrapeSodaImp extends SodaImp {
   GrapeSodaImp() {}

   public void pourSodaImp() {
       System.out.println("Delicious Grape Soda!");
   }
}
```

OrangeSodaImp

```
package pk.dydakt.to.dp.s.gof.bridge;

public class OrangeSodaImp extends SodaImp {
   OrangeSodaImp() {}
```

```
    public void pourSodaImp() {
        System.out.println("Citrusy Orange Soda!");
    }
}
```

Wyjście na konsoli:

```
testing medium soda on the cherry platform
...glug...Yummy Cherry Soda!
...glug...Yummy Cherry Soda!

testing super size soda on the cherry platform
...glug...Yummy Cherry Soda!
...glug...Yummy Cherry Soda!
...glug...Yummy Cherry Soda!
...glug...Yummy Cherry Soda!
...glug...Yummy Cherry Soda!

testing medium soda on the grape platform
...glug...Delicious Grape Soda!
...glug...Delicious Grape Soda!

testing super size soda on the grape platform
...glug...Delicious Grape Soda!
...glug...Delicious Grape Soda!
...glug...Delicious Grape Soda!
...glug...Delicious Grape Soda!
...glug...Delicious Grape Soda!

testing medium soda on the orange platform
...glug...Citrusy Orange Soda!
...glug...Citrusy Orange Soda!

testing super size soda on the orange platform
...glug...Citrusy Orange Soda!
...glug...Citrusy Orange Soda!
...glug...Citrusy Orange Soda!
...glug...Citrusy Orange Soda!
...glug...Citrusy Orange Soda!
```

5. Facade

Przeznaczenie (cytat z książki GoF):
Zapewnia jednolity interfejs dla podsystemu zawierającego wiele interfejsów. Definiuje interfejs wyższego poziomu, co ułatwia korzystanie z podsystemu.
Przeznaczenie (cytat z fluffycat):
Jedna klasa zawiera metodę realizującą złożony proces wywołań metod innych klas.

Opis: Przykład pokazuje, w jaki sposób klient restauracji nie troszcząc się o szczegóły może skierować zamówienie na herbatę do kelnera. Prosi on kelnera o zrobienie filiżanki herbaty. Korzysta jedynie z ogólnego opisu tego, czego oczekuje, czyli właśnie z interfejsu Facade. To implementacja tego interfejsu, którą może być zespół osób składających się z osoby odpowiedzialnej za przygotowanie filiżanki, za znalezienie torebki herbaty, za przygotowanie wrzątku, za zalanie torebki herbaty wrzątkiem współpracuje ze sobą komunikując się wzajemnie w sposób niewidoczny dla klienta fasady zajmuje się realizacją ogólnego zamówienia klienta na poziomie bardziej szczegółowym. Można powiedzieć, że ten zespół osób skrywa się za fasadą ogólnego zamówienia.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_S_GoF_Facade

```
package pk.dydakt.to.dp.s.gof.facade;

public class _DP_S_GoF_Facade {

   public static void main(String[] args) {
       FacadeCuppaMaker cuppaMaker = new FacadeCuppaMaker();
       FacadeTeaCup teaCup = cuppaMaker.makeACuppa();
       System.out.println(teaCup);
   }
}
```

Klasy wzorca:

FacadeWater

```
package pk.dydakt.to.dp.s.gof.facade;

public class FacadeWater {
   boolean waterIsBoiling;

   public FacadeWater() {
       setWaterIsBoiling(false);
       System.out.println("behold the wonderous water");
   }

   public void boilFacadeWater() {
       setWaterIsBoiling(true);
       System.out.println("water is boiling");
   }

   public void setWaterIsBoiling(boolean isWaterBoiling) {
       waterIsBoiling = isWaterBoiling;
   }
   public boolean getWaterIsBoiling() {
```

```
        return waterIsBoiling;
    }
}
```

FacadeTeaBag

```
package pk.dydakt.to.dp.s.gof.facade;

public class FacadeTeaBag {
    public FacadeTeaBag() {
        System.out.println("behold the lovely tea bag");
    }
}
```

FacadeTeaCup

```
package pk.dydakt.to.dp.s.gof.facade;

public class FacadeTeaCup {
    boolean teaBagIsSteeped;
    FacadeWater facadeWater;
    FacadeTeaBag facadeTeaBag;

    public FacadeTeaCup() {
        setTeaBagIsSteeped(false);
        System.out.println("behold the beautiful new tea cup");
    }

    public void setTeaBagIsSteeped(boolean isTeaBagSteeped) {
        teaBagIsSteeped = isTeaBagSteeped;
    }
    public boolean getTeaBagIsSteeped() {
        return teaBagIsSteeped;
    }

    public void addFacadeTeaBag(FacadeTeaBag facadeTeaBagIn) {
        facadeTeaBag = facadeTeaBagIn;
        System.out.println("the tea bag is in the tea cup");
    }

    public void addFacadeWater(FacadeWater facadeWaterIn) {
        facadeWater = facadeWaterIn;
        System.out.println("the water is in the tea cup");
    }

    public void steepTeaBag() {
        if ( (facadeTeaBag != null) &&
              ( (facadeWater != null) &&
                (facadeWater.getWaterIsBoiling()) )
           ) {
           System.out.println("the tea is steeping in the cup");
           setTeaBagIsSteeped(true);
        } else {
           System.out.println("the tea is not steeping in the cup");
           setTeaBagIsSteeped(false);
        }
    }

    public String toString() {
        if (this.getTeaBagIsSteeped()) {
            return ("A nice cuppa tea!");
        } else {
```

```
            String tempString = new String("A cup with ");
            if (facadeWater != null) {
                 if (facadeWater.getWaterIsBoiling()) {
                     tempString = (tempString + "boiling water ");
                 } else {
                     tempString = (tempString + "cold water ");
                 }
            } else {
                 tempString = (tempString + "no water ");
            }

            if (facadeTeaBag != null) {
                 tempString = (tempString + "and a tea bag");
            } else {
                 tempString = (tempString + "and no tea bag");
            }
            return tempString;
        }

    }
}
```

FacadeCuppaMaker

```
package pk.dydakt.to.dp.s.gof.facade;

public class FacadeCuppaMaker {
   boolean teaBagIsSteeped;

   public FacadeCuppaMaker() {
       System.out.println(
         "FacadeCuppaMaker ready to make you a cuppa!");
   }

   public FacadeTeaCup makeACuppa() {
       FacadeTeaCup cup = new FacadeTeaCup();
       FacadeTeaBag teaBag = new FacadeTeaBag();
       FacadeWater water = new FacadeWater();
       cup.addFacadeTeaBag(teaBag);
       water.boilFacadeWater();
       cup.addFacadeWater(water);
       cup.steepTeaBag();
       return cup;
   }
}
```

Wyjście na konsoli:

```
FacadeCuppaMaker ready to make you a cuppa!
behold the beautiful new tea cup
behold the lovely tea bag
behold the wonderous water
the tea bag is in the tea cup
water is boiling
the water is in the tea cup
the tea is steeping in the cup
A nice cuppa tea!
```

6. Proxy

Przeznaczenie (cytat z książki GoF):
Zapewnia substytut lub reprezentanta innego obiektu w celu sterowania dostępem do niego.
Przeznaczenie (cytat z fluffycat):
Jedna klasa steruje tworzeniem i dostępem do obiektów innej klasy.

Opis: Klient w restauracji zamawia filiżankę herbaty. Zamawia ją u kelnera. Kelner przekazuje żądanie do osoby zajmującej się parzeniem herbaty. Sam się nie zajmuje jednak realizacją zamówienia. Wydaje jedynie polecenia: weź filiżankę i zaparz w niej herbatę. Kelner działa tu jak proxy.
Sens tworzenia proxy z samej filiżanki herbaty jest dla mnie niejasny.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_S_GoF_Proxy

```
package pk.dydakt.to.dp.s.gof.proxy;

public class _DP_S_GoF_Proxy {

   public static void main(String[] args) {
       System.out.println("TestProxy: instantiating PotOfTeaProxy");
       PotOfTeaInterface potOfTea = new PotOfTeaProxy();
       System.out.println(" ");
       System.out.println("TestProxy: pouring tea");
       potOfTea.pourTea();
   }
}
```

Klasy wzorca:

PotOfTeaInterface

```
package pk.dydakt.to.dp.s.gof.proxy;

//PotOfTeaInterface will insure that the proxy
//has the same methods as it's real subject
public interface PotOfTeaInterface {
   public void pourTea();
}
```

PotOfTea

```
package pk.dydakt.to.dp.s.gof.proxy;

public class PotOfTea implements PotOfTeaInterface {
   public PotOfTea() {
       System.out.println("Making a pot of tea");
   }

   public void pourTea() {
       System.out.println("Pouring tea");
   }
}
```

PotOfTeaProxy

```
package pk.dydakt.to.dp.s.gof.proxy;

public class PotOfTeaProxy implements PotOfTeaInterface {
   PotOfTea potOfTea;

   public PotOfTeaProxy() {}

   public void pourTea() {
       potOfTea = new PotOfTea();
       potOfTea.pourTea();
   }
}
```

Wyjście na konsoli:

```
TestProxy: instantiating PotOfTeaProxy

TestProxy: pouring tea
Making a pot of tea
Pouring tea
```

7. Flyweight

Przeznaczenie (cytat z książki GoF):
Wykorzystuje współdzielenie obiektów w celu efektywnej obsługi wielkiej ilości drobnych obiektów.
Przeznaczenie (cytat z fluffycat):
Część ponownie używana oraz część zmienna klasy są rozbijane na dwie klasy aby oszczędzić zasoby.

Opis: Kelnerzy zbierają zamówienia od klientów siedzących przy różnych stolikach. Przekazują je do kuchni, która realizuje zamówienia klientów. Lista zamawianych smaków herbat jest niewielka w porównaniu z ilością zamówień. Czyli, mówiąc wprost jest to wielka ale niezbyt wyrafinowana restauracja. Pyłkiem jest tu smak herbaty, ale nie numer stolika. Za tworzenie smaków odpowiada fabryka smaków herbaty!? Jeśli w zamówieniu pojawi się nowy smak, to fabryka smaków jest o niego wzbogacana i smak ten może być wielokrotnie wykorzystany przy zamówieniach spływających z wielu stolików.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_S_GoF_Flyweight

```
package pk.dydakt.to.dp.s.gof.flyweight;

public class _DP_S_GoF_Flyweight {

   static TeaFlavor[] flavors =
     new TeaFlavor[100];
     //the flavors ordered
   static TeaOrderContext[] tables =
     new TeaOrderContext[100];
     //the tables for the orders
   static int ordersMade = 0;
   static TeaFlavorFactory teaFlavorFactory;

   static void takeOrders(String flavorIn, int table) {
       flavors[ordersMade] =
         teaFlavorFactory.getTeaFlavor(flavorIn);
       tables[ordersMade++] =
         new TeaOrderContext(table);
   }

   public static void main(String[] args) {
       teaFlavorFactory = new TeaFlavorFactory();

       takeOrders("chai", 2);
       takeOrders("chai", 2);
       takeOrders("camomile", 1);
       takeOrders("camomile", 1);
       takeOrders("earl grey", 1);
       takeOrders("camomile", 897);
       takeOrders("chai", 97);
       takeOrders("chai", 97);
       takeOrders("camomile", 3);
       takeOrders("earl grey", 3);
```

```
        takeOrders("chai", 3);
        takeOrders("earl grey", 96);
        takeOrders("camomile", 552);
        takeOrders("chai", 121);
        takeOrders("earl grey", 121);

        for (int i = 0; i < ordersMade; i++) {
            flavors[i].serveTea(tables[i]);
        }
        System.out.println(" ");
        System.out.println("total teaFlavor objects made: " +
                           teaFlavorFactory.getTotalTeaFlavorsMade());
    }
}
```

Klasy wzorca:

TeaOrder

```
package pk.dydakt.to.dp.s.gof.flyweight;

public abstract class TeaOrder {
    public abstract void serveTea(TeaOrderContext teaOrderContext);
}
```

TeaOrderContext

```
package pk.dydakt.to.dp.s.gof.flyweight;

public class TeaOrderContext {
   int tableNumber;

   TeaOrderContext(int tableNumber) {
       this.tableNumber = tableNumber;
   }

   public int getTable() {
       return this.tableNumber;
   }
}
```

TeaFlavor

```
package pk.dydakt.to.dp.s.gof.flyweight;

public class TeaFlavor extends TeaOrder {
    String teaFlavor;

    TeaFlavor(String teaFlavor) {
        this.teaFlavor = teaFlavor;
    }

    public String getFlavor() {
        return this.teaFlavor;
    }

    public void serveTea(TeaOrderContext teaOrderContext) {
        System.out.println("Serving tea flavor " +
                           teaFlavor +
                          " to table number " +
                           teaOrderContext.getTable());
    }
}
```

TeaFlavorFactory

```
package pk.dydakt.to.dp.s.gof.flyweight;

public class TeaFlavorFactory {
   TeaFlavor[] flavors = new TeaFlavor[10];
     //no more than 10 flavors can be made
   int teasMade = 0;

   public TeaFlavor getTeaFlavor(String flavorToGet) {
       if (teasMade > 0) {
           for (int i = 0; i < teasMade; i++) {
               if (flavorToGet.equals((flavors[i]).getFlavor())) {
                   return flavors[i];
               }
           }
       }
       flavors[teasMade] = new TeaFlavor(flavorToGet);
       return flavors[teasMade++];
   }

   public int getTotalTeaFlavorsMade() {return teasMade;}
}
```

Wyjście na konsoli:

```
Serving tea flavor chai to table number 2
Serving tea flavor chai to table number 2
Serving tea flavor camomile to table number 1
Serving tea flavor camomile to table number 1
Serving tea flavor earl grey to table number 1
Serving tea flavor camomile to table number 897
Serving tea flavor chai to table number 97
Serving tea flavor chai to table number 97
Serving tea flavor camomile to table number 3
Serving tea flavor earl grey to table number 3
Serving tea flavor chai to table number 3
Serving tea flavor earl grey to table number 96
Serving tea flavor camomile to table number 552
Serving tea flavor chai to table number 121
Serving tea flavor earl grey to table number 121

total teaFlavor objects made: 3
```

Wzorce czynnościowe

Bazują one na następującej dziedzinie wiedzy i systemie pojęć:

(...diagram)

1. Observer

Przeznaczenie (cytat z książki GoF):
Określa zależność jeden-do-wiele między obiektami. Gdy jeden z nich zmienia swój stan, wszystkie obiekty od niego zależne są o tym automatycznie powiadamiane i uaktualniane.
Przeznaczenie (cytat z fluffycat):
Obiekt informuje inne obiekty o swojej zmianie.

Opis: W przykładzie mamy DVD należące do różnych kategorii. Mamy też subscriberów oczekujących na odbiór zamówionych przez siebie DVD. Tak naprawde subskrybują oni określone kategorie DVD a nie konkretne tytuły. W pewnym momencie na rynku (albo u dostawcy) ukazują się nowe tytuły należące do tych kategorii, na które oczekują subskrybenci. Subskrybenci są obserwatorami rynku. Ukazanie się nowego tytułu sprawia, że są oni powiadamiani o ukzaniu się interesującego ich tytułu.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_Observer

```
package pk.dydakt.to.dp.b.gof.observer;

public class _DP_B_GoF_Observer {

   public static void main(String[] args) {
       DvdReleaseByCategory btvs =
           new DvdReleaseByCategory("Buffy the Vampire Slayer");
       DvdReleaseByCategory simpsons =
           new DvdReleaseByCategory("The Simpsons");
       DvdReleaseByCategory sopranos =
           new DvdReleaseByCategory("The Sopranos");
       DvdReleaseByCategory xfiles =
           new DvdReleaseByCategory("The X-Files");

       DvdSubscriber jsopra = new DvdSubscriber("Junior Soprano");
       DvdSubscriber msimps = new DvdSubscriber("Maggie Simpson");
       DvdSubscriber rgiles = new DvdSubscriber("Rupert Giles");
       DvdSubscriber smulde = new DvdSubscriber("Samantha Mulder");
       DvdSubscriber wrosen = new DvdSubscriber("Willow Rosenberg");

       btvs.addSubscriber(rgiles);
       btvs.addSubscriber(wrosen);
       simpsons.addSubscriber(msimps);
       sopranos.addSubscriber(jsopra);
       xfiles.addSubscriber(smulde);
       xfiles.addSubscriber(wrosen);

       DvdRelease btvsS2 =
         new DvdRelease("DVDFOXBTVSS20",
                        "Buffy The Vampire Slayer Season 2",
                         2002, 06, 11);
       DvdRelease simpS2 =
         new DvdRelease("DVDFOXSIMPSO2",
                        "The Simpsons Season 2",
                         2002, 07,  9);
       DvdRelease soprS2 =
```

```
        new DvdRelease("DVDHBOSOPRAS2",
                       "The Sopranos Season 2",
                        2001, 11,  6);
      DvdRelease xfilS5 =
        new DvdRelease("DVDFOXXFILES5",
                       "The X-Files Season 5",
                        2002, 04,  1);

      btvs.newDvdRelease(btvsS2);
      simpsons.newDvdRelease(simpS2);
      sopranos.newDvdRelease(soprS2);
      xfiles.newDvdRelease(xfilS5);

      xfiles.removeSubscriber(wrosen);

      xfilS5.updateDvdReleaseDate(2002, 5, 14);
      xfiles.updateDvd(xfilS5);
   }
}
```

Klasy wzorca:

DvdSubscriber

```
package pk.dydakt.to.dp.b.gof.observer;

public class DvdSubscriber {
   private String subscriberName;

   public DvdSubscriber(String subscriberNameIn) {
       this.subscriberName = subscriberNameIn;
   }

   public String getSubscriberName() {
       return this.subscriberName;
   }

   public void newDvdRelease(DvdRelease newDvdRelease,
                             String subscriptionListName) {
       System.out.println("");
       System.out.println("Hello " + this.getSubscriberName() +
                          ", subscriber to the " +
                             subscriptionListName +
                          " DVD release list.");
       System.out.println("The new Dvd " +
                            newDvdRelease.getDvdName() +
                          " will be released on " +
                            newDvdRelease.getDvdReleaseMonth() + "/" +
                            newDvdRelease.getDvdReleaseDay() + "/" +
                            newDvdRelease.getDvdReleaseYear() + ".");
   }

   public void updateDvdRelease(DvdRelease newDvdRelease,
                                String subscriptionListName) {
       System.out.println("");
       System.out.println("Hello " + this.getSubscriberName() +
                          ", subscriber to the " +
                            subscriptionListName +
                          " DVD release list.");
       System.out.println(
         "The following DVDs release has been revised: " +
         newDvdRelease.getDvdName() + " will be released on " +
         newDvdRelease.getDvdReleaseMonth() + "/" +
```

```
            newDvdRelease.getDvdReleaseDay() + "/" +
            newDvdRelease.getDvdReleaseYear() + ".");
    }
}
```

DvdRelease

```
package pk.dydakt.to.dp.b.gof.observer;

public class DvdRelease {
    private String serialNumber;
    private String dvdName;
    private int dvdReleaseYear;
    private int dvdReleaseMonth;
    private int dvdReleaseDay;


    public DvdRelease(String serialNumber,
                      String dvdName,
                      int dvdReleaseYear,
                      int dvdReleaseMonth,
                      int dvdReleaseDay) {
        setSerialNumber(serialNumber);
        setDvdName(dvdName);
        setDvdReleaseYear(dvdReleaseYear);
        setDvdReleaseMonth(dvdReleaseMonth);
        setDvdReleaseDay(dvdReleaseDay);
    }

    public void updateDvdRelease(String serialNumber,
                                 String dvdName,
                                 int dvdReleaseYear,
                                 int dvdReleaseMonth,
                                 int dvdReleaseDay) {
        setSerialNumber(serialNumber);
        setDvdName(dvdName);
        setDvdReleaseYear(dvdReleaseYear);
        setDvdReleaseMonth(dvdReleaseMonth);
        setDvdReleaseDay(dvdReleaseDay);
    }

    public void updateDvdReleaseDate(int dvdReleaseYear,
                                     int dvdReleaseMonth,
                                     int dvdReleaseDay) {
        setDvdReleaseYear(dvdReleaseYear);
        setDvdReleaseMonth(dvdReleaseMonth);
        setDvdReleaseDay(dvdReleaseDay);
    }

    public void setSerialNumber(String serialNumberIn) {
        this.serialNumber = serialNumberIn;
    }
    public String getSerialNumber() {
        return this.serialNumber;
    }

    public void setDvdName(String dvdNameIn) {
        this.dvdName = dvdNameIn;
    }
    public String getDvdName() {
        return this.dvdName;
    }
```

```
    public void setDvdReleaseYear(int dvdReleaseYearIn) {
        this.dvdReleaseYear = dvdReleaseYearIn;
    }
    public int getDvdReleaseYear() {
        return this.dvdReleaseYear;
    }

    public void setDvdReleaseMonth(int dvdReleaseMonthIn) {
        this.dvdReleaseMonth = dvdReleaseMonthIn;
    }
    public int getDvdReleaseMonth() {
        return this.dvdReleaseMonth;
    }

    public void setDvdReleaseDay(int dvdReleaseDayIn) {
        this.dvdReleaseDay = dvdReleaseDayIn;
    }
    public int getDvdReleaseDay() {
        return this.dvdReleaseDay;
    }
}
```

DvdReleaseByCategory

```
package pk.dydakt.to.dp.b.gof.observer;

import java.util.ArrayList;
import java.util.ListIterator;

public class DvdReleaseByCategory {
    String categoryName;
    ArrayList<DvdSubscriber> subscriberList = new
ArrayList<DvdSubscriber>();
    ArrayList<DvdRelease> dvdReleaseList = new ArrayList<DvdRelease>();

    public DvdReleaseByCategory(String categoryNameIn) {
        categoryName = categoryNameIn;
    }

    public String getCategoryName() {
        return this.categoryName;
    }

    public boolean addSubscriber(DvdSubscriber dvdSubscriber) {
        return subscriberList.add(dvdSubscriber);
    }

    public boolean removeSubscriber(DvdSubscriber dvdSubscriber) {
        ListIterator listIterator = subscriberList.listIterator();
        while (listIterator.hasNext()) {
            if (dvdSubscriber == (DvdSubscriber)(listIterator.next())) {
                listIterator.remove();
                return true;
            }
        }
        return false;
    }

    public void newDvdRelease(DvdRelease dvdRelease) {
        dvdReleaseList.add(dvdRelease);
        notifySubscribersOfNewDvd(dvdRelease);
```

```
    }

    public void updateDvd(DvdRelease dvdRelease) {
        boolean dvdUpdated = false;
        DvdRelease tempDvdRelease;
        ListIterator<DvdRelease> listIterator =
dvdReleaseList.listIterator();
        while (listIterator.hasNext()) {
            tempDvdRelease = (DvdRelease)listIterator.next();
            if (dvdRelease.getSerialNumber().
                  equals(tempDvdRelease.getSerialNumber())) {
                listIterator.remove();
                listIterator.add(dvdRelease);
                dvdUpdated = true;
                break;
            }
        }
        if (dvdUpdated == true) {
            notifySubscribersOfUpdate(dvdRelease);
        } else {
            this.newDvdRelease(dvdRelease);
        }
    }

    private void notifySubscribersOfNewDvd(DvdRelease dvdRelease) {
        ListIterator listIterator = subscriberList.listIterator();
        while (listIterator.hasNext()) {
            ((DvdSubscriber)(listIterator.next())).
                newDvdRelease(dvdRelease, this.getCategoryName());
        }
    }

    private void notifySubscribersOfUpdate(DvdRelease dvdRelease) {
        ListIterator listIterator = subscriberList.listIterator();
        while (listIterator.hasNext()) {
            ((DvdSubscriber)(listIterator.next())).
                updateDvdRelease(dvdRelease, this.getCategoryName() );
        }
    }
}
```

Wyjście na konsoli:

```
Hello Rupert Giles, subscriber to the Buffy the Vampire Slayer DVD release list.
The new Dvd Buffy The Vampire Slayer Season 2 will be released on 6/11/2002.

Hello Willow Rosenberg, subscriber to the Buffy the Vampire Slayer DVD release list.
The new Dvd Buffy The Vampire Slayer Season 2 will be released on 6/11/2002.

Hello Maggie Simpson, subscriber to the The Simpsons DVD release list.
The new Dvd The Simpsons Season 2 will be released on 7/9/2002.

Hello Junior Soprano, subscriber to the The Sopranos DVD release list.
The new Dvd The Sopranos Season 2 will be released on 11/6/2001.

Hello Samantha Mulder, subscriber to the The X-Files DVD release list.
The new Dvd The X-Files Season 5 will be released on 4/1/2002.

Hello Willow Rosenberg, subscriber to the The X-Files DVD release list.
The new Dvd The X-Files Season 5 will be released on 4/1/2002.

Hello Samantha Mulder, subscriber to the The X-Files DVD release list.
The following DVDs release has been revised: The X-Files Season 5 will be released on
5/14/2002.
```

2. Visitor

Przeznaczenie (cytat z książki GoF):
Określa operację, która ma być wykonana na elementach struktury obiektowej. Umożliwia definiowanie nowej operacji bez modyfikowania klas elementów, na których ona działa.
Przeznaczenie (cytat z fluffycat):
Jedna lub więcej klas pozostających we wzajemnych relacjach mają tę samą metodę, która wywołuje metodę specyficzną dla nich w innej klasie.

Opis: Co to jest blurb? Tworzone są tu zarówno krótkie jak i długie opisy na DVD. Każde z DVD może być grą, książką albo filmem. Są też dwaj odwiedzający: jeden dla krótkich opisów, a drugi dla długich. Każdy z nich odwiedza klasy konkretne z hierarchii informacji o korzeniu w klasie AbstractTitleInfo. Przykładem może być system tworzący opisy płytek DVD na podstawie dostępnych dla niego informacji o tytułach i o charakterze nagrania.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_Visitor

```
package pk.dydakt.to.dp.b.gof.visitor;

public class _DP_B_GoF_Visitor {

   public static void main(String[] args) {
       AbstractTitleInfo bladeRunner =
         new DvdInfo("Blade Runner", "Harrison Ford", '1');
       AbstractTitleInfo electricSheep =
         new BookInfo("Do Androids Dream of Electric Sheep?",
                      "Phillip K. Dick");
       AbstractTitleInfo sheepRaider =
         new GameInfo("Sheep Raider");

       TitleBlurbVisitor titleLongBlurbVisitor =
         new TitleLongBlurbVisitor();

       System.out.println("Long Blurbs:");
       bladeRunner.accept(titleLongBlurbVisitor);
       System.out.println("Testing bladeRunner   " +
                          titleLongBlurbVisitor.getTitleBlurb());
       electricSheep.accept(titleLongBlurbVisitor);
       System.out.println("Testing electricSheep " +
                          titleLongBlurbVisitor.getTitleBlurb());
       sheepRaider.accept(titleLongBlurbVisitor);
       System.out.println("Testing sheepRaider   " +
                          titleLongBlurbVisitor.getTitleBlurb());

       TitleBlurbVisitor titleShortBlurbVisitor =
         new TitleShortBlurbVisitor();

       System.out.println("Short Blurbs:");
       bladeRunner.accept(titleShortBlurbVisitor);
       System.out.println("Testing bladeRunner   " +
         titleShortBlurbVisitor.getTitleBlurb());
       electricSheep.accept(titleShortBlurbVisitor);
       System.out.println("Testing electricSheep " +
```

```
          titleShortBlurbVisitor.getTitleBlurb());
        sheepRaider.accept(titleShortBlurbVisitor);
        System.out.println("Testing sheepRaider   " +
          titleShortBlurbVisitor.getTitleBlurb());
    }
}
```

Klasy wzorca:

AbstractTitleInfo

```
package pk.dydakt.to.dp.b.gof.visitor;

public abstract class AbstractTitleInfo {
    private String titleName;
    public final void setTitleName(String titleNameIn) {
        this.titleName = titleNameIn;
    }
    public final String getTitleName() {
        return this.titleName;
    }

    public abstract void accept(TitleBlurbVisitor titleBlurbVisitor);
}
```

BookInfo

```
package pk.dydakt.to.dp.b.gof.visitor;

public class BookInfo extends AbstractTitleInfo {
    private String author;

    public BookInfo(String titleName, String author) {
        this.setTitleName(titleName);
        this.setAuthor(author);
    }

    public void setAuthor(String authorIn) {
        this.author = authorIn;
    }
    public String getAuthor() {
        return this.author;
    }

    public void accept(TitleBlurbVisitor titleBlurbVisitor) {
        titleBlurbVisitor.visit(this);
    }
}
```

DvdInfo

```
package pk.dydakt.to.dp.b.gof.visitor;

public class DvdInfo extends AbstractTitleInfo {
    private String star;
    private char encodingRegion;

    public DvdInfo(String titleName,
                   String star,
                   char encodingRegion) {
        this.setTitleName(titleName);
        this.setStar(star);
        this.setEncodingRegion(encodingRegion);
    }
```

```
    public void setStar(String starIn) {
        this.star = starIn;
    }
    public String getStar() {
        return this.star;
    }
    public void setEncodingRegion(char encodingRegionIn) {
        this.encodingRegion = encodingRegionIn;
    }
    public char getEncodingRegion() {
        return this.encodingRegion;
    }

    public void accept(TitleBlurbVisitor titleBlurbVisitor) {
        titleBlurbVisitor.visit(this);
    }
}
```

GameInfo

```
package pk.dydakt.to.dp.b.gof.visitor;

public class GameInfo extends AbstractTitleInfo {
    public GameInfo(String titleName) {
        this.setTitleName(titleName);
    }

    public void accept(TitleBlurbVisitor titleBlurbVisitor) {
        titleBlurbVisitor.visit(this);
    }
}
```

TitleBlurbVisitor

```
package pk.dydakt.to.dp.b.gof.visitor;

public abstract class TitleBlurbVisitor {
    String titleBlurb;
    public void setTitleBlurb(String blurbIn) {
        this.titleBlurb = blurbIn;
    }
    public String getTitleBlurb() {
        return this.titleBlurb;
    }

    public abstract void visit(BookInfo bookInfo);
    public abstract void visit(DvdInfo dvdInfo);
    public abstract void visit(GameInfo gameInfo);
}
```

TitleLongBlurbVisitor

```
package pk.dydakt.to.dp.b.gof.visitor;

public class TitleLongBlurbVisitor extends TitleBlurbVisitor {
    public void visit(BookInfo bookInfo) {
        this.setTitleBlurb("LB-Book: " +
                           bookInfo.getTitleName() +
                           ", Author: " +
                           bookInfo.getAuthor());
    }
```

```
    public void visit(DvdInfo dvdInfo) {
        this.setTitleBlurb("LB-DVD: " +
                           dvdInfo.getTitleName() +
                           ", starring " +
                           dvdInfo.getStar() +
                           ", encoding region: " +
                           dvdInfo.getEncodingRegion());
    }

    public void visit(GameInfo gameInfo) {
        this.setTitleBlurb("LB-Game: " +
                           gameInfo.getTitleName());
    }
}
```

TitleShortBlurbVisitor

```
package pk.dydakt.to.dp.b.gof.visitor;

public class TitleShortBlurbVisitor extends TitleBlurbVisitor {
    public void visit(BookInfo bookInfo) {
        this.setTitleBlurb("SB-Book: " + bookInfo.getTitleName());
    }

    public void visit(DvdInfo dvdInfo) {
        this.setTitleBlurb("SB-DVD: " + dvdInfo.getTitleName());
    }

    public void visit(GameInfo gameInfo) {
        this.setTitleBlurb("SB-Game: " + gameInfo.getTitleName());
    }
}
```

Wyjście na konsoli:

```
Long Blurbs:
Testing bladeRunner   LB-DVD: Blade Runner, starring Harrison Ford, encoding region: 1
Testing electricSheep LB-Book: Do Androids Dream of Electric Sheep?, Author: Phillip K. Dick
Testing sheepRaider   LB-Game: Sheep Raider
Short Blurbs:
Testing bladeRunner   SB-DVD: Blade Runner
Testing electricSheep SB-Book: Do Androids Dream of Electric Sheep?
Testing sheepRaider   SB-Game: Sheep Raider
```

3. Command

Przeznaczenie (cytat z książki GoF):
Kapsułkuje żądania w postaci obiektu, co umożliwia parametryzowanie klientów różnymi żądaniami, kolejkowanie żądań lub zapisywanie ich w dziennikach, a także ułatwia implementację anulowanych operacji.
Przeznaczenie (cytat z fluffycat):
Obiekt kapsułkuje wszystko, co jest potrzebne do wykonania metody w innym obiekcie.

Opis: Dodawanie, a nastepnie usuwanie (anulowanie dodania) gwiazdek zamiast spacji pomiędzy wyrazami w tytule DVD. Przykładem może być narzędzie do tworzenia opisów płytek DVD, które pokazuje użytkownikowi ostatnio zastosowany efekt. Użytkownik może albo zatwierdzić ten efekt i przejść do ewentualnego następnego albo może go odrzucić i ewentualnie zastosować inny efekt.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_Command

```
package pk.dydakt.to.dp.b.gof.command;

public class _DP_B_GoF_Command {

   public static void main(String[] args) {
       DvdName jayAndBob =
         new DvdName("Jay and Silent Bob Strike Back");
       DvdName spongeBob =
         new DvdName("Sponge Bob Squarepants - " +
                     "Nautical Nonsense and Sponge Buddies");
       System.out.println("as first instantiated");
       System.out.println(jayAndBob.toString());
       System.out.println(spongeBob.toString());

       CommandAbstract bobStarsOn =
         new DvdCommandNameStarsOn(jayAndBob);
       CommandAbstract bobStarsOff =
         new DvdCommandNameStarsOff(jayAndBob);
       CommandAbstract spongeStarsOn =
         new DvdCommandNameStarsOn(spongeBob);
       CommandAbstract spongeStarsOff =
         new DvdCommandNameStarsOff(spongeBob);

       bobStarsOn.execute();
       spongeStarsOn.execute();
       System.out.println(" ");
       System.out.println("stars on");
       System.out.println(jayAndBob.toString());
       System.out.println(spongeBob.toString());

       spongeStarsOff.execute();
       System.out.println(" ");
       System.out.println("sponge stars off");
       System.out.println(jayAndBob.toString());
       System.out.println(spongeBob.toString());
   }
```

```
}
```

Klasy wzorca:

DvdName

```
package pk.dydakt.to.dp.b.gof.command;

public class DvdName {
   private String titleName;

   public DvdName(String titleName) {
       this.setTitleName(titleName);
   }

   public final void setTitleName(String titleNameIn) {
       this.titleName = titleNameIn;
   }
   public final String getTitleName() {
       return this.titleName;
   }

   public void setNameStarsOn() {
       this.setTitleName(this.getTitleName().replace(' ','*'));
   }
   public void setNameStarsOff() {
       this.setTitleName(this.getTitleName().replace('*',' '));
   }

   public String toString() {
       return ("DVD: " + this.getTitleName());
   }
}
```

CommandAbstract

```
package pk.dydakt.to.dp.b.gof.command;

public abstract class CommandAbstract {
   public abstract void execute();
}
```

DvdCommandNameStarsOff

```
package pk.dydakt.to.dp.b.gof.command;

public class DvdCommandNameStarsOff extends CommandAbstract {
   private DvdName dvdName;

   public DvdCommandNameStarsOff(DvdName dvdNameIn) {
       this.dvdName = dvdNameIn;
   }
   public void execute() {
       this.dvdName.setNameStarsOff();
   }
}
```

DvdCommandNameStarsOn

```
package pk.dydakt.to.dp.b.gof.command;

public class DvdCommandNameStarsOn extends CommandAbstract {
   private DvdName dvdName;

   public DvdCommandNameStarsOn(DvdName dvdNameIn) {
```

```
        this.dvdName = dvdNameIn;
    }
    public void execute() {
        this.dvdName.setNameStarsOn();
    }
}
```

Wyjście na konsoli:

```
as first instantiated
DVD: Jay and Silent Bob Strike Back
DVD: Sponge Bob Squarepants - Nautical Nonsense and Sponge Buddies

stars on
DVD: Jay*and*Silent*Bob*Strike*Back
DVD: Sponge*Bob*Squarepants*-*Nautical*Nonsense*and*Sponge*Buddies

sponge stars off
DVD: Jay*and*Silent*Bob*Strike*Back
DVD: Sponge Bob Squarepants - Nautical Nonsense and Sponge Buddies
```

4. Strategy

Przeznaczenie (cytat z książki GoF):
Definiuje rodzinę algorytmów, kapsułkuje każdy z nich i umożliwia ich wymianę. Sprawia też, że możliwe staje się zmienianie algorytmu niezależnie od używających go klientów.
Przeznaczenie (cytat z fluffycat):
Obiekt decyduje o tym, która rodzina metod zostanie wywoływana. Każda metoda jest w innej własnej klasie rozszerzającej wspólną dla nich klasę bazową.

Opis: Różne sposoby formatowania tytułów. Przykład może ilustrować zastosowanie systemu softwerowego do tworzenia różnych wersji tytułów. Zastosowane strategie modyfikowania pierwotnej wersji tytułu mogą być wykorzystywane np. do tworzenia danych ułatwiających wyszukiwanie tytułów w bazie danych lub sortowanie wg istotnych słów z tytułu, co zresztą też może być przydatne przy wyszukiwaniu.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_Strategy

```
package pk.dydakt.to.dp.b.gof.strategy;

public class _DP_B_GoF_Strategy {

   public static void main(String[] args) {
       DvdNameContext allCapContext =
           new DvdNameContext('C');
       DvdNameContext theEndContext =
           new DvdNameContext('E');
       DvdNameContext spacesContext =
           new DvdNameContext('S');

       String dvdNames[] = new String[3];
       dvdNames[0] = "Jay and Silent Bob Strike Back";
       dvdNames[1] = "The Fast and the Furious";
       dvdNames[2] = "The Others";

       char replaceChar = '*';

       System.out.println("Testing formatting with all caps");
       String[] dvdCapNames =
           allCapContext.formatDvdNames(dvdNames);

       System.out.println(" ");
       System.out.println(
           "Testing formatting with beginning the at end");
       String[] dvdEndNames =
           theEndContext.formatDvdNames(dvdNames);

       System.out.println(" ");
       System.out.println(
           "Testing formatting with all spaces replaced with " +
           replaceChar);
       String[] dvdSpcNames =
           spacesContext.formatDvdNames(dvdNames, replaceChar);
   }
```

```
}
```

Klasy wzorca:

DvdNameContext

```
package pk.dydakt.to.dp.b.gof.strategy;

public class DvdNameContext {
   private DvdNameStrategy dvdNameStrategy;

   public DvdNameContext(char strategyTypeIn) {
       switch (strategyTypeIn) {
           case 'C' :
             this.dvdNameStrategy = new DvdNameAllCapStrategy();
             break;
           case 'E' :
             this.dvdNameStrategy = new DvdNameTheAtEndStrategy();
             break;
           case 'S' :
             this.dvdNameStrategy =
               new DvdNameReplaceSpacesStrategy();
             break;
           default  :
             this.dvdNameStrategy = new DvdNameTheAtEndStrategy();
       }
   }

   public String[] formatDvdNames(String[] namesIn) {
       return this.formatDvdNames(namesIn, ' ');
   }

   public String[] formatDvdNames(String[] namesIn, char replacementIn) {
       String[] namesOut = new String[namesIn.length];
       for (int i = 0; i < namesIn.length; i++) {
           namesOut[i] =
             dvdNameStrategy.formatDvdName(namesIn[i], replacementIn);
           System.out.println(
             "Dvd name before formatting: " + namesIn[i]);
           System.out.println(
             "Dvd name after formatting:  " + namesOut[i]);
           System.out.println("==========================");
       }
       return namesOut;
   }
}
```

DvdNameStrategy

```
package pk.dydakt.to.dp.b.gof.strategy;

public abstract class DvdNameStrategy {
   public abstract String
     formatDvdName(String dvdName, char charIn);
}
```

DvdNameAllCapStrategy

```
package pk.dydakt.to.dp.b.gof.strategy;

public class DvdNameAllCapStrategy extends DvdNameStrategy {
   public String formatDvdName(String dvdName, char charIn) {
       return dvdName.toUpperCase();
```

```
    }
}
```

DvdNameReplaceSpacesStrategy

```
package pk.dydakt.to.dp.b.gof.strategy;

public class DvdNameReplaceSpacesStrategy extends DvdNameStrategy {
   public String formatDvdName(String dvdName, char charIn) {
       return dvdName.replace(' ', charIn);
   }
}
```

DvdNameTheAtEndStrategy

```
package pk.dydakt.to.dp.b.gof.strategy;

public class DvdNameTheAtEndStrategy extends DvdNameStrategy {
   public String formatDvdName(String dvdName, char charIn) {
       if (dvdName.startsWith("The ")) {
           return new String(dvdName.substring(4,
             (dvdName.length())) + ", The");
       }
       if (dvdName.startsWith("THE ")) {
           return new String(dvdName.substring(4,
             (dvdName.length())) + ", THE");
       }
       if (dvdName.startsWith("the ")) {
           return new String(dvdName.substring(4,
             (dvdName.length())) + ", the");
       }
       return dvdName;
   }
}
```

Wyjście na konsoli:

```
Testing formatting with all caps
Dvd name before formatting: Jay and Silent Bob Strike Back
Dvd name after formatting:  JAY AND SILENT BOB STRIKE BACK
==========================
Dvd name before formatting: The Fast and the Furious
Dvd name after formatting:  THE FAST AND THE FURIOUS
==========================
Dvd name before formatting: The Others
Dvd name after formatting:  THE OTHERS
==========================

Testing formatting with beginning the at end
Dvd name before formatting: Jay and Silent Bob Strike Back
Dvd name after formatting:  Jay and Silent Bob Strike Back
==========================
Dvd name before formatting: The Fast and the Furious
Dvd name after formatting:  Fast and the Furious, The
==========================
Dvd name before formatting: The Others
Dvd name after formatting:  Others, The
==========================

Testing formatting with all spaces replaced with *
Dvd name before formatting: Jay and Silent Bob Strike Back
Dvd name after formatting:  Jay*and*Silent*Bob*Strike*Back
==========================
Dvd name before formatting: The Fast and the Furious
Dvd name after formatting:  The*Fast*and*the*Furious
==========================
Dvd name before formatting: The Others
Dvd name after formatting:  The*Others
==========================
```

5. Iterator

Przeznaczenie (cytat z książki GoF):
Zapewnia sekwencyjny dostęp do elementów obiektu zagregowanego bez ujawniania jego reprezentacji wewnętrznej.
Przeznaczenie (cytat z fluffycat):
Jeden obiekt może przejść przez elementy innego obiektu.

Opis: Lista tytułów DVD oraz iterator przechodzacy po tej liście i wyświetlający wszystkie tytuły znajdujące się na niej w trakcie iterowania. Pomiędzy iterowaniem mogą być robione modyfikacje listy polegające na usuwaniu, zmienianiu lub dodawaniu tytułów. Wszystkie te zmianu są widoczne w następującej po nich iteracji.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_Iterator

```
package pk.dydakt.to.dp.b.gof.iterator;

public class _DP_B_GoF_Iterator {

   public static void main(String[] args) {
       DvdList fiveShakespeareMovies = new DvdList();
       fiveShakespeareMovies.append("10 Things I Hate About You");
       fiveShakespeareMovies.append("Shakespeare In Love");
       fiveShakespeareMovies.append("O (2001)");
       fiveShakespeareMovies.append("American Pie 2");
       fiveShakespeareMovies.append("Scotland, PA.");
       fiveShakespeareMovies.append("Hamlet (2000)");

       DvdListIterator fiveShakespeareIterator =
         fiveShakespeareMovies.createIterator();
       while (!fiveShakespeareIterator.isDone()) {
           System.out.println(fiveShakespeareIterator.currentItem());
           fiveShakespeareIterator.next();
       }

       fiveShakespeareMovies.delete("American Pie 2");

       System.out.println(" ");
       fiveShakespeareIterator.first();
       while (!fiveShakespeareIterator.isDone()) {
           System.out.println(fiveShakespeareIterator.currentItem());
           fiveShakespeareIterator.next();
       }
   }
}
```

Klasy wzorca:

DvdList

```
package pk.dydakt.to.dp.b.gof.iterator;

public class DvdList {
```

```
    private String[] titles;
    //Yes, it would be easier to do this whole example with ArrayList
    // and ListIterator, but it certainly wouldn't be as much fun!
    private int titleCount;
    //title count is always a real count of titles, but one ahead of
    //itself as a subscript
    private int arraySize;

    public DvdList() {
         titles = new String[3];
         //using 3 to demonstrate array expansion more easily,
         //  not for efficency
         titleCount = 0;
         arraySize = 3;
    }

    public int count() {
        return titleCount;
    }

    public void append(String titleIn) {
        if (titleCount >= arraySize) {
            String[] tempArray = new String[arraySize];
            for (int i = 0; i < arraySize; i++)
            {tempArray[i] = titles[i];}
            titles = null;
            arraySize = arraySize + 3;
            titles = new String[arraySize];
            for (int i = 0; i < (arraySize - 3); i++) {
                titles[i] = tempArray[i];
            }
        }
        titles[titleCount++] = titleIn;
    }

    public void delete(String titleIn) {
        boolean found = false;
        for (int i = 0; i < (titleCount -1); i++) {
            if (found == false) {
                if (titles[i].equals(titleIn)) {
                    found = true;
                    titles[i] = titles[i + 1];
                }
            } else {
                if (i < (titleCount -1)) {
                    titles[i] = titles[i + 1];
                } else {
                    titles[i] = null;
                }
            }
        }

        if (found == true) {
            --titleCount;
        }
    }


    public DvdListIterator createIterator() {
        return new InnerIterator();
    }
```

```
   //note:
   //  This example shows the Concrete Iterator as an inner class.
   //  The Iterator Pattern in GoF does allow for multiple types of
   //  iterators for a given list or "Aggregate".  This could be
   //  accomplished with multiple Iterators in multiple inner classes.
   //  The createIterator class would then have multiple variations.
   //  This, however, assumes that you will have a limited number of
   //  iterator variants - which is normally the case.  If you do want
   //  more flexibility in iterator creation, the iterators should not
   //  be in inner classes, and perhaps some sort factory should be
   //  employed to create them.
   //
   private class InnerIterator implements DvdListIterator {
       private int currentPosition = 0;

       private InnerIterator() {}

       public void first() {
           currentPosition = 0;
       }

       public void next() {
           if (currentPosition < (titleCount)) {
               ++currentPosition;
           }
       }

       public boolean isDone() {
           if (currentPosition >= (titleCount)) {
               return true;
           } else {
               return false;
           }
       }

       public String currentItem() {
           return titles[currentPosition];
       }
   }
}
```

DvdListIterator

```
package pk.dydakt.to.dp.b.gof.iterator;

public interface DvdListIterator {
   public void first();
   public void next();
   public boolean isDone();
   public String currentItem();
}
```

Wyjście na konsoli:

```
10 Things I Hate About You
Shakespeare In Love
O (2001)
American Pie 2
Scotland, PA.
Hamlet (2000)

10 Things I Hate About You
```

```
Shakespeare In Love
O (2001)
Scotland, PA.
Hamlet (2000)
```

6. Template Method

Przeznaczenie (cytat z książki GoF):
Definiuje szkielet algorytmu jako operację, odkładając definicję niektórych kroków algorytmu do podklas. Umożliwia podklasom przedefiniowywanie pewnych kroków algorytmu bez zmiany jego struktury.
Przeznaczenie (cytat z fluffycat):
Klasa abstrakcyjna definiuje różne metody oraz zawiera jedną nie nadpisywaną w jej podklasach metodę, która wywołuje różne metody.

Opis: Co to jest blurb? Przykład ilustruje sposób podpisyania płytek DVD oparty na ogólnym algorytmie wspólnym dla wszystkich rodzajów nagrań (książki, filmy,...). Algorytm ten jednak w pewnym swoim fragmencie odwołuje się do operacji specyficznych dla każdego z rodzaju nagrania. Zdaje się w tym momencie na swoje podklasy.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_Template

```
package pk.dydakt.to.dp.b.gof.template;

public class _DP_B_GoF_Template {

   public static void main(String[] args) {
       TitleInfo bladeRunner =
         new DvdTitleInfo("Blade Runner",
                          "Harrison Ford", '1');
       TitleInfo electricSheep =
         new BookTitleInfo("Do Androids Dream of Electric Sheep?",
                           "Phillip K. Dick");
       TitleInfo sheepRaider =
         new GameTitleInfo("Sheep Raider");

       System.out.println(" ");
       System.out.println("Testing bladeRunner   " +
                          bladeRunner.ProcessTitleInfo());
       System.out.println("Testing electricSheep " +
                          electricSheep.ProcessTitleInfo());
       System.out.println("Testing sheepRaider   " +
                          sheepRaider.ProcessTitleInfo());
   }
}
```

Klasy wzorca:

TitleInfo

```
package pk.dydakt.to.dp.b.gof.template;

public abstract class TitleInfo {
   private String titleName;

   //the "template method" -
   //  calls the concrete class methods, is not overridden
   public final String ProcessTitleInfo() {
```

```
        StringBuffer titleInfo = new StringBuffer();

        titleInfo.append(this.getTitleBlurb());
        titleInfo.append(this.getDvdEncodingRegionInfo());

        return titleInfo.toString();
    }

    //the following 2 methods are "concrete abstract class methods"
    public final void setTitleName(String titleNameIn) {
        this.titleName = titleNameIn;
    }
    public final String getTitleName() {
        return this.titleName;
    }

    //this is a "primitive operation",
    //  and must be overridden in the concrete templates
    public abstract String getTitleBlurb();

    //this is a "hook operation", which may be overridden,
    //hook operations usually do nothing if not overridden
    public String getDvdEncodingRegionInfo() {
        return " ";
    }
}
```

BookTitleInfo

```
package pk.dydakt.to.dp.b.gof.template;

public class BookTitleInfo extends TitleInfo {
    private String author;

    public BookTitleInfo(String titleName, String author) {
        this.setTitleName(titleName);
        this.setAuthor(author);
    }

    public void setAuthor(String authorIn) {this.author = authorIn;}
    public String getAuthor() {return this.author;}

    public String getTitleBlurb() {
        return ("Book: " + this.getTitleName() +
                ", Author: " + this.getAuthor());
    }
}
```

DvdTitleInfo

```
package pk.dydakt.to.dp.b.gof.template;

public class DvdTitleInfo extends TitleInfo {
    private String star;
    private char encodingRegion;

    public DvdTitleInfo(String titleName,
                        String star,
                        char encodingRegion) {
        this.setTitleName(titleName);
        this.setStar(star);
        this.setEncodingRegion(encodingRegion);
```

```
    }

    public void setStar(String starIn) {
        this.star = starIn;
    }
    public String getStar() {
        return this.star;
    }
    public void setEncodingRegion(char encodingRegionIn) {
        this.encodingRegion = encodingRegionIn;
    }
    public char getEncodingRegion() {
        return this.encodingRegion;
    }

    public String getTitleBlurb() {
        return ("DVD: " + this.getTitleName() +
                ", starring " + this.getStar());
    }

    public String getDvdEncodingRegionInfo() {
        return (", encoding region: " + this.getEncodingRegion());
    }
}
```

GameTitleInfo

```
package pk.dydakt.to.dp.b.gof.template;

public class GameTitleInfo extends TitleInfo {
    public GameTitleInfo(String titleName) {
        this.setTitleName(titleName);
    }

    public String getTitleBlurb() {
        return ("Game: " + this.getTitleName());
    }
}
```

Wyjście na konsoli:

```
Testing bladeRunner   DVD: Blade Runner, starring Harrison Ford, encoding region: 1
Testing electricSheep Book: Do Androids Dream of Electric Sheep?, Author: Phillip K. Dick
Testing sheepRaider   Game: Sheep Raider
```

7. State

Przeznaczenie (cytat z książki GoF):
Umożliwia obiektowi zmianę zachowania, gdy zmienia się jego stan wewnętrzny. Dzięki temu obiekt zdaje się zmieniać wówczas swoją klasę.
Przeznaczenie (cytat z fluffycat):
Obiekt wydaje się zmieniać swoją klasę, gdy klasa, poprzez którą przekazuje wywołania ...
(An object appears to change its` class when the class it passes calls through to switches itself for a related class.)

Opis: Przykład pokazuje, w jaki sposób można osiągnąć efekt sekwencji zmian stanu obiketu w sposób niewidoczny dla klienta wzorca. Rozwiązania tego można użyć np. do podpisywania w ustalonym cyklu płytek DVD z tytułami tworzonymi wg różnych konwencji. Może to być przydatne, jeśli chcemy podpisywać DVD seriami dla klientów zamawiających podobne zestawy płytek. Chodzi nam wtedy o to, aby całe zestawy płytek były dostarczane do klienta najszybciej jak to tylkomożliwe. Klient, który zapisał się pierwszy dostaje swój zestaw jako pierwszy z ewentualną modyfikacją polegającą na uwzględnieniu dostwaw najpierw dla większych grup odbiorców. Co zrobić, jeśli będziemy chcieli zmienić zawartość zestawu DVD? - można sparametryzować zmiany stanu.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_State

```
package pk.dydakt.to.dp.b.gof.state;

public class _DP_B_GoF_State {

   public static void main(String[] args)
   {
       DvdStateContext stateContext = new DvdStateContext();
       stateContext.showName(
         "Sponge Bob Squarepants - "+
           "Nautical Nonsense and Sponge Buddies");
       stateContext.showName(
         "Jay and Silent Bob Strike Back");
       stateContext.showName(
         "Buffy The Vampire Slayer Season 2");
       stateContext.showName(
         "The Sopranos Season 2");
   }
}
```

Klasy wzorca:

DvdStateContext

```
package pk.dydakt.to.dp.b.gof.state;

public class DvdStateContext {
   private DvdStateName dvdStateName;

   public DvdStateContext() {
       setDvdStateName(new DvdStateNameStars());
```

```
        //start with stars
    }

    public void setDvdStateName(DvdStateName dvdStateNameIn) {
        this.dvdStateName = dvdStateNameIn;
    }

    public void showName(String nameIn) {
        this.dvdStateName.showName(this, nameIn);
    }
}
```

DvdStateName

```
package pk.dydakt.to.dp.b.gof.state;

public interface DvdStateName {
   public void showName(DvdStateContext dvdStateContext,
                        String nameIn);
}
```

DvdStateNameExclaim

```
package pk.dydakt.to.dp.b.gof.state;

public class DvdStateNameExclaim implements DvdStateName {
    public DvdStateNameExclaim() {}

    public void showName(DvdStateContext dvdStateContext,
                         String nameIn) {
         System.out.println(nameIn.replace(' ','!'));
         //show exclaim only once, switch back to stars
         dvdStateContext.setDvdStateName(new DvdStateNameStars());
    }
}
```

DvdStateNameStars

```
package pk.dydakt.to.dp.b.gof.state;

public class DvdStateNameStars implements DvdStateName {
    int starCount;

    public DvdStateNameStars() {
        starCount = 0;
    }

    public void showName(DvdStateContext dvdStateContext,
                         String nameIn) {
         System.out.println(nameIn.replace(' ','*'));
         // show stars twice, switch to exclamation point
         if (++starCount > 1) {
             dvdStateContext.setDvdStateName(
               new DvdStateNameExclaim());
         }
    }
}
```

Wyjście na konsoli:

```
Sponge*Bob*Squarepants*-*Nautical*Nonsense*and*Sponge*Buddies
Jay*and*Silent*Bob*Strike*Back
Buffy!The!Vampire!Slayer!Season!2
The*Sopranos*Season*2
```

8. Mediator

Przeznaczenie (cytat z książki GoF):
Definuje obiekt kapsułkujący informacje o tym, jak obiekty współdziałają. Przyczynia się do rozluźnienia powiązań między obiektami, gdyż sprawia, że nie odwołują się one do siebie wprost. Umożliwia też oddzielne zmienianie ich sposobu porozumiewania się.
Przeznaczenie (cytat z fluffycat):
Obiekt dystrybuuje komunikację pomiędzy wieloma obiektami.

Opis: Przykład pokazuje różne wersje formatowania nazwiska wraz z tytułem. Może to być wymuszone np. przez różne konwencje obowiązujące w różnych krajach. Widać również, że zależności pomiędzy klasami reprezentującymi różne wersje formatowania zostały uproszczone dzięki temu, że komunikują się one pomiędzy sobą (muszą się komunikować) za pośrednictwem mediatora przekazywanego do nich w konstruktorze.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_Mediator

```
package pk.dydakt.to.dp.b.gof.mediator;

public class _DP_B_GoF_Mediator {

   public static void main(String[] args) {
       DvdMediator dvdMediator = new DvdMediator();
       DvdLowercaseTitle dvdLower =
         new DvdLowercaseTitle("Mulholland Dr.", dvdMediator);
       DvdUpcaseTitle dvdUp =
         new DvdUpcaseTitle(dvdLower, dvdMediator);

       System.out.println("Lowercase LC title :" +
                          dvdLower.getLowercaseTitle());
       System.out.println("Lowercase super title :" +
                          dvdLower.getTitle());
       System.out.println("Upcase UC title :" +
                          dvdUp.getUpcaseTitle());
       System.out.println("Upcase super title :" +
                          dvdUp.getTitle());

       dvdLower.setSuperTitleLowercase();

       System.out.println(" ");
       System.out.println("After Super set to LC");
       System.out.println("Lowercase LC title :" +
                          dvdLower.getLowercaseTitle());
       System.out.println("Lowercase super title :" +
                          dvdLower.getTitle());
       System.out.println("Upcase UC title :" +
                          dvdUp.getUpcaseTitle());
       System.out.println("Upcase super title :" +
                          dvdUp.getTitle());
   }
}
```

Klasy wzorca:

DvdTitle

```
package pk.dydakt.to.dp.b.gof.mediator;

public abstract class DvdTitle {
   private String title;

   public void setTitle(String titleIn) {
       this.title = titleIn;
   }
   public String getTitle() {
       return this.title;
   }
}
```

DvdLowercaseTitle

```
package pk.dydakt.to.dp.b.gof.mediator;

public class DvdLowercaseTitle extends DvdTitle {
   private String LowercaseTitle;
   private DvdMediator dvdMediator;

   public DvdLowercaseTitle(String title, DvdMediator dvdMediator) {
       this.setTitle(title);
       resetTitle();
       this.dvdMediator = dvdMediator;
       this.dvdMediator.setLowercase(this);
   }

   public DvdLowercaseTitle(DvdTitle dvdTitle,
                            DvdMediator dvdMediator) {
       this(dvdTitle.getTitle(), dvdMediator);
   }

   public void resetTitle() {
       this.setLowercaseTitle(this.getTitle().toLowerCase());
   }
   public void resetTitle(String title) {
       this.setTitle(title);
       this.resetTitle();
   }

   public void setSuperTitleLowercase() {
       this.setTitle(this.getLowercaseTitle());
       dvdMediator.changeTitle(this);
   }

   public String getLowercaseTitle() {
       return LowercaseTitle;
   }
   private void setLowercaseTitle(String LowercaseTitle) {
       this.LowercaseTitle = LowercaseTitle;
   }
}
```

DvdUpcaseTitle

```
package pk.dydakt.to.dp.b.gof.mediator;

public class DvdUpcaseTitle extends DvdTitle {
```

```
    private String upcaseTitle;
    private DvdMediator dvdMediator;

    public DvdUpcaseTitle(String title,
                          DvdMediator dvdMediator) {
        this.setTitle(title);
        resetTitle();
        this.dvdMediator = dvdMediator;
        this.dvdMediator.setUpcase(this);
    }

    public DvdUpcaseTitle(DvdTitle dvdTitle,
                          DvdMediator dvdMediator) {
        this(dvdTitle.getTitle(), dvdMediator);
    }

    public void resetTitle() {
        this.setUpcaseTitle(this.getTitle().toUpperCase());
    }
    public void resetTitle(String title) {
        this.setTitle(title);
        this.resetTitle();
    }

    public void setSuperTitleUpcase() {
        this.setTitle(this.getUpcaseTitle());
        dvdMediator.changeTitle(this);
    }

    public String getUpcaseTitle() {
       return upcaseTitle;
    }
    private void setUpcaseTitle(String upcaseTitle) {
        this.upcaseTitle = upcaseTitle;
    }
}
```

DvdMediator

```
package pk.dydakt.to.dp.b.gof.mediator;

public class DvdMediator {
    private DvdUpcaseTitle dvdUpcaseTitle;
    private DvdLowercaseTitle dvdLowercaseTitle;

    public void setUpcase(DvdUpcaseTitle dvdUpcaseTitle) {
        this.dvdUpcaseTitle = dvdUpcaseTitle;
    }

    public void setLowercase(DvdLowercaseTitle dvdLowercaseTitle) {
        this.dvdLowercaseTitle = dvdLowercaseTitle;
    }

    public void changeTitle(DvdUpcaseTitle dvdUpcaseTitle) {
        this.dvdLowercaseTitle.resetTitle(dvdUpcaseTitle.getTitle());
    }

    public void changeTitle(DvdLowercaseTitle dvdLowercaseTitle) {
        this.dvdUpcaseTitle.resetTitle(dvdLowercaseTitle.getTitle());
    }
}
```

Wyjście na konsoli:

```
Lowercase LC title :mulholland dr.
Lowercase super title :Mulholland Dr.
Upcase UC title :MULHOLLAND DR.
Upcase super title :Mulholland Dr.

After Super set to LC
Lowercase LC title :mulholland dr.
Lowercase super title :mulholland dr.
Upcase UC title :MULHOLLAND DR.
Upcase super title :mulholland dr.
```

9. Chain of Responsiblity

Przeznaczenie (cytat z książki GoF):
Umożliwia uniknięcie związania wysyłającego żądanie z odbiorcą żądania przez danie więcej niż jednemu obiektowi szansy obsłużenia tego żądania. Tworzy łańcuch odbierających obiektów i przekazuje wzdłuż niego żądanie, aż jakiś obiekt je obsłuży.
Przeznaczenie (cytat z fluffycat):
Metoda wołana w jednej klasie może przejść w górę hierarchii aby znaleźć obiket, który może poprawnie wykonać tę metodę.

Opis: Przykład pokazuje mechanizm budowania tytułu z uwzglednieniem kategorii filmu. Na początku budowany jest łańcuch obiektów odpowiedzialnych za poźniejsze prawidłowe zbudowanie tytułu.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_ChainOfResponsibility

```
package pk.dydakt.to.dp.b.gof.chainofresponsibility;

public class _DP_B_GoF_ChainOfResponsibility {

   public static void main(String[] args) {
        String topTitle;
        DvdCategory comedy = new DvdCategory("Comedy");
        comedy.setTopCategoryTitle("Ghost World");

        DvdSubCategory comedyChildrens =
          new DvdSubCategory(comedy, "Childrens");

        DvdSubSubCategory comedyChildrensAquatic =
          new DvdSubSubCategory(comedyChildrens, "Aquatic");
        comedyChildrensAquatic.setTopSubSubCategoryTitle(
          "Sponge Bob Squarepants");

        System.out.println("");
        System.out.println("Getting top comedy title:");
        topTitle = comedy.getTopTitle();
        System.out.println("The top title for " +
                           comedy.getAllCategories() +
                           " is " + topTitle);

        System.out.println("");
        System.out.println("Getting top comedy/childrens title:");
        topTitle = comedyChildrens.getTopTitle();
        System.out.println("The top title for " +
                           comedyChildrens.getAllCategories() +
                           " is " + topTitle);

        System.out.println("");
        System.out.println(
          "Getting top comedy/childrens/aquatic title:");
        topTitle = comedyChildrensAquatic.getTopTitle();
        System.out.println("The top title for " +
                           comedyChildrensAquatic.getAllCategories() +
```

```
                                " is " + topTitle);
    }
}
```

Klasy wzorca:

TopTitle

```
package pk.dydakt.to.dp.b.gof.chainofresponsibility;

public interface TopTitle {
   public String getTopTitle();

   public String getAllCategories();
}
```

DvdCategory

```
package pk.dydakt.to.dp.b.gof.chainofresponsibility;

public class DvdCategory implements TopTitle {
   private String category;
   private String topCategoryTitle;

   public DvdCategory(String category) {
        this.setCategory(category);
   }

   public void setCategory(String categoryIn) {
       this.category = categoryIn;
   }
   public String getCategory() {
       return this.category;
   }
   public String getAllCategories() {
       return getCategory();
   }

   public void setTopCategoryTitle(String topCategoryTitleIn) {
       this.topCategoryTitle = topCategoryTitleIn;
   }
   public String getTopCategoryTitle() {
       return this.topCategoryTitle;
   }

   public String getTopTitle() {
       return this.topCategoryTitle;
   }
}
```

DvdSubCategory

```
package pk.dydakt.to.dp.b.gof.chainofresponsibility;

public class DvdSubCategory implements TopTitle {
   private String subCategory;
   private String topSubCategoryTitle;
   private DvdCategory parent;

   public DvdSubCategory(DvdCategory dvdCategory, String subCategory) {
        this.setSubCategory(subCategory);
        this.parent = dvdCategory;
   }
```

```
    public void setSubCategory(String subCategoryIn) {
        this.subCategory = subCategoryIn;
    }
    public String getSubCategory() {
        return this.subCategory;
    }
    public void setCategory(String categoryIn) {
        parent.setCategory(categoryIn);
    }
    public String getCategory() {
        return parent.getCategory();
    }
    public String getAllCategories() {
        return (getCategory() + "/" + getSubCategory());
    }

    public void setTopSubCategoryTitle(String topSubCategoryTitleIn) {
        this.topSubCategoryTitle = topSubCategoryTitleIn;
    }
    public String getTopSubCategoryTitle() {
        return this.topSubCategoryTitle;
    }
    public void setTopCategoryTitle(String topCategoryTitleIn) {
        parent.setTopCategoryTitle(topCategoryTitleIn);
    }
    public String getTopCategoryTitle() {
        return parent.getTopCategoryTitle();
    }

    public String getTopTitle() {
        if (null != getTopSubCategoryTitle()) {
            return this.getTopSubCategoryTitle();
        } else {
            System.out.println("no top title in Category/SubCategory " +
                               getAllCategories());
            return parent.getTopTitle();
        }
    }
}
```

DvdSubSubCategory

```
package pk.dydakt.to.dp.b.gof.chainofresponsibility;

public class DvdSubSubCategory implements TopTitle {
    private String subSubCategory;
    private String topSubSubCategoryTitle;
    private DvdSubCategory parent;

    public DvdSubSubCategory(DvdSubCategory dvdSubCategory,
                             String subCategory) {
         this.setSubSubCategory(subCategory);
         this.parent = dvdSubCategory;
    }

    public void setSubSubCategory(String subSubCategoryIn) {
        this.subSubCategory = subSubCategoryIn;
    }
    public String getSubSubCategory() {
        return this.subSubCategory;
    }
```

```
    public void setSubCategory(String subCategoryIn) {
        parent.setSubCategory(subCategoryIn);
    }
    public String getSubCategory() {
        return parent.getSubCategory();
    }
    public void setCategory(String categoryIn) {
        parent.setCategory(categoryIn);
    }
    public String getCategory() {
        return parent.getCategory();
    }
    public String getAllCategories() {
        return (getCategory() + "/" +
                getSubCategory() + "/" +
                getSubSubCategory());}

    public void setTopSubSubCategoryTitle(
      String topSubSubCategoryTitleIn) {
       this.topSubSubCategoryTitle = topSubSubCategoryTitleIn;
    }
    public String getTopSubSubCategoryTitle() {
        return this.topSubSubCategoryTitle;
    }
    public void setTopSubCategoryTitle(
      String topSubCategoryTitleIn) {
        parent.setTopSubCategoryTitle(topSubCategoryTitleIn);
    }
    public String getTopSubCategoryTitle() {
        return parent.getTopSubCategoryTitle();
    }
    public void setTopCategoryTitle(String topCategoryTitleIn) {
        parent.setTopCategoryTitle(topCategoryTitleIn);
    }
    public String getTopCategoryTitle() {
        return parent.getTopCategoryTitle();
    }

    public String getTopTitle() {
        if (null != getTopSubSubCategoryTitle()) {
            return this.getTopSubSubCategoryTitle();
        } else {
            System.out.println(
              "no top title in Category/SubCategory/SubSubCategory " +
               getAllCategories());
            return parent.getTopTitle();
        }
    }
}
```

Wyjście na konsoli:

```
Getting top comedy title:
The top title for Comedy is Ghost World

Getting top comedy/childrens title:
no top title in Category/SubCategory Comedy/Childrens
The top title for Comedy/Childrens is Ghost World

Getting top comedy/childrens/aquatic title:
The top title for Comedy/Childrens/Aquatic is Sponge Bob Squarepants
```

10. Interpreter

Przeznaczenie (cytat z książki GoF):
Definiuje reprezentację dla gramatyki zadanego języka, a także interpreter, który wykorzystuje tę reprezentację do interpretowania zdań w zadanym języku.
Przeznaczenie (cytat z fluffycat):
Definiuje jezyk i jego syntaktykę, dokonując analizy syntaktycznej wejścia na obiektach mogących realizować wymagane operacje.

Opis: Przykład ilustruje sposób budowania zapytań. Zapytania te są najpierw budowane w odpowiednim języku, a następnie są interpretowane. W czasie interpretowania ich tworzona jest odpowiedź na te zapytania.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_Interpreter

```
package pk.dydakt.to.dp.b.gof.interpreter;

public class _DP_B_GoF_Interpreter {

   public static void main(String[] args) {
       DvdInterpreterContext dvdInterpreterContext =
         new DvdInterpreterContext();
       dvdInterpreterContext.addTitle("Caddy Shack");
       dvdInterpreterContext.addTitle("Training Day");
       dvdInterpreterContext.addTitle("Hamlet");

       dvdInterpreterContext.addActor("Ethan Hawke");
       dvdInterpreterContext.addActor("Denzel Washington");

       dvdInterpreterContext.addTitleAndActor(
         new TitleAndActor("Hamlet", "Ethan Hawke"));
       dvdInterpreterContext.addTitleAndActor(
         new TitleAndActor("Training Day", "Ethan Hawke"));
       dvdInterpreterContext.addTitleAndActor(
         new TitleAndActor("Caddy Shack", "Ethan Hawke"));
       dvdInterpreterContext.addTitleAndActor(
         new TitleAndActor("Training Day", "Denzel Washington"));

       DvdInterpreterClient dvdInterpreterClient =
         new DvdInterpreterClient(dvdInterpreterContext);

       System.out.println(
         "interpreting show actor: " +
          dvdInterpreterClient.interpret(
            "show actor"));
       System.out.println(
          "interpreting show actor for title : " +
           dvdInterpreterClient.interpret("show actor for title <Training
Day>"));
       System.out.println(
          "interpreting show title: " +
           dvdInterpreterClient.interpret(
             "show title"));
```

```
        System.out.println(
           "interpreting show title for actor : " +
           dvdInterpreterClient.interpret("show title for actor <Ethan
Hawke>"));
    }
}
```

Klasy wzorca:

TitleAndActor

```
package pk.dydakt.to.dp.b.gof.interpreter;

public class TitleAndActor {
    private String title;
    private String actor;
    public TitleAndActor(String title, String actor) {
        this.title = title;
        this.actor = actor;
    }
    public String getTitle() {return this.title;}
    public String getActor() {return this.actor;}
}
```

DvdInterpreterClient

```
package pk.dydakt.to.dp.b.gof.interpreter;

import java.util.StringTokenizer;

public class DvdInterpreterClient {
   DvdInterpreterContext dvdInterpreterContext;

   public DvdInterpreterClient(
     DvdInterpreterContext dvdInterpreterContext) {
       this.dvdInterpreterContext = dvdInterpreterContext;
   }

   // expression syntax:
   // show title | actor [for actor | title ]
   public String interpret(String expression) {
       StringBuffer result = new StringBuffer("Query Result: ");

       String currentToken;
       StringTokenizer expressionTokens =
         new StringTokenizer(expression);

       char mainQuery = ' ';
       char subQuery = ' ';
       boolean forUsed = false;
       String searchString = null;
       boolean searchStarted = false;
       boolean searchEnded = false;

       while (expressionTokens.hasMoreTokens())
       {
           currentToken = expressionTokens.nextToken();
           if (currentToken.equals("show")) {
               continue;
               //show in all queries, not really used
           } else if (currentToken.equals("title")) {
               if (mainQuery == ' ') {
```

```
                    mainQuery = 'T';
                } else {
                    if ((subQuery == ' ') && (forUsed)) {
                        subQuery = 'T';
                    }
                }
            } else if (currentToken.equals("actor")) {
                if (mainQuery == ' ') {
                    mainQuery = 'A';
                } else {
                    if ((subQuery == ' ') && (forUsed)) {
                        subQuery = 'A';
                    }
                }
            } else if (currentToken.equals("for")) {
                forUsed = true;
            } else if ((searchString == null) &&
                       (subQuery != ' ') &&
                       (currentToken.startsWith("<"))) {
                  searchString = currentToken;
                  searchStarted = true;
                  if (currentToken.endsWith(">")) {
                      searchEnded = true;
                  }
            } else if ((searchStarted) && (!searchEnded)) {
                searchString = searchString + " " + currentToken;
                if (currentToken.endsWith(">")) {
                    searchEnded = true;
                }
            }
        }

        if (searchString != null) {
            searchString =
              searchString.substring(1,(searchString.length() - 1));
              //remove <>
        }

        DvdAbstractExpression abstractExpression;

        switch (mainQuery) {
            case 'A' :  {
                switch (subQuery) {
                    case 'T' : {
                        abstractExpression =
                          new DvdActorTitleExpression(searchString);
                        break;
                    }
                    default : {
                        abstractExpression =
                          new DvdActorExpression();
                        break;
                    }
                }
                break;
            }
            case 'T' : {
                switch (subQuery) {
                    case 'A' : {
                        abstractExpression =
                          new DvdTitleActorExpression(searchString);
```

```
                        break;
                    }
                    default : {
                        abstractExpression = new DvdTitleExpression();
                        break;
                    }
                }
                break;
            }

            default : return result.toString();
        }


        result.append(
          abstractExpression.interpret(dvdInterpreterContext));

        return result.toString();
    }
}
```

DvdAbstractExpression

```
package pk.dydakt.to.dp.b.gof.interpreter;

public abstract class DvdAbstractExpression {
   public abstract String interpret(
     DvdInterpreterContext dvdInterpreterContext);
}
```

DvdActorExpression

```
package pk.dydakt.to.dp.b.gof.interpreter;

import java.util.ArrayList;
import java.util.ListIterator;

public class DvdActorExpression extends DvdAbstractExpression {
   public String interpret(DvdInterpreterContext dvdInterpreterContext) {
        ArrayList actors = dvdInterpreterContext.getAllActors();
        ListIterator actorsIterator = actors.listIterator();
        StringBuffer titleBuffer = new StringBuffer("");
        boolean first = true;
        while (actorsIterator.hasNext()) {
            if (!first) {
                titleBuffer.append(", ");
            } else {
                first = false;
            }
            titleBuffer.append((String)actorsIterator.next());
        }
        return titleBuffer.toString();
    }
}
```

DvdActorTitleExpression

```
package pk.dydakt.to.dp.b.gof.interpreter;

import java.util.ArrayList;
import java.util.ListIterator;

public class DvdActorTitleExpression extends DvdAbstractExpression {
```

```
    String title;

    public DvdActorTitleExpression(String title) {
        this.title = title;
    }

    public String interpret(DvdInterpreterContext dvdInterpreterContext) {
        ArrayList actorsAndTitles =
            dvdInterpreterContext.getActorsForTitle(title);
        ListIterator actorsAndTitlesIterator =
            actorsAndTitles.listIterator();
        StringBuffer actorBuffer = new StringBuffer("");
        boolean first = true;
        while (actorsAndTitlesIterator.hasNext()) {
            if (!first) {
                actorBuffer.append(", ");
            } else {
                first = false;
            }
            actorBuffer.append((String)actorsAndTitlesIterator.next());
        }
        return actorBuffer.toString();
    }
}
```

DvdInterpreterContext

```
package pk.dydakt.to.dp.b.gof.interpreter;

import java.util.ArrayList;
import java.util.ListIterator;

public class DvdInterpreterContext {
    private ArrayList<String> titles = new ArrayList<String>();
    private ArrayList<String> actors = new ArrayList<String>();
    private ArrayList<TitleAndActor> titlesAndActors = new
ArrayList<TitleAndActor>();

    public void addTitle(String title) {
        titles.add(title);
    }
    public void addActor(String actor) {
        actors.add(actor);
    }
    public void addTitleAndActor(TitleAndActor titleAndActor) {
        titlesAndActors.add(titleAndActor);
    }

    public ArrayList getAllTitles() {
        return titles;
    }
    public ArrayList getAllActors() {
        return actors;
    }
    public ArrayList<String> getActorsForTitle(String titleIn) {
        ArrayList<String> actorsForTitle = new ArrayList<String>();
        TitleAndActor tempTitleAndActor;
        ListIterator titlesAndActorsIterator =
          titlesAndActors.listIterator();
        while (titlesAndActorsIterator.hasNext()) {
            tempTitleAndActor =
              (TitleAndActor)titlesAndActorsIterator.next();
```

```
            if (titleIn.equals(tempTitleAndActor.getTitle())) {
                actorsForTitle.add(tempTitleAndActor.getActor());
            }
        }
        return actorsForTitle;
    }
    public ArrayList<String> getTitlesForActor(String actorIn) {
        ArrayList<String> titlesForActor = new ArrayList<String>();
        TitleAndActor tempTitleAndActor;
        ListIterator actorsAndTitlesIterator =
          titlesAndActors.listIterator();
        while (actorsAndTitlesIterator.hasNext()) {
            tempTitleAndActor =
              (TitleAndActor)actorsAndTitlesIterator.next();
            if (actorIn.equals(tempTitleAndActor.getActor())) {
                titlesForActor.add(tempTitleAndActor.getTitle());
            }
        }
        return titlesForActor;
    }
}
```

DvdTitleActorExpression

```
package pk.dydakt.to.dp.b.gof.interpreter;

import java.util.ArrayList;
import java.util.ListIterator;

public class DvdTitleActorExpression extends DvdAbstractExpression {
    String title;

    public DvdTitleActorExpression(String title) {
        this.title = title;
    }

    public String interpret(DvdInterpreterContext dvdInterpreterContext) {
        ArrayList titlesAndActors =
            dvdInterpreterContext.getTitlesForActor(title);
        ListIterator titlesAndActorsIterator =
            titlesAndActors.listIterator();
        StringBuffer titleBuffer = new StringBuffer("");
        boolean first = true;
        while (titlesAndActorsIterator.hasNext()) {
            if (!first) {
                titleBuffer.append(", ");
            } else {
                first = false;
            }
            titleBuffer.append((String)titlesAndActorsIterator.next());
        }
        return titleBuffer.toString();
    }
}
```

DvdTitleExpression

```
package pk.dydakt.to.dp.b.gof.interpreter;

import java.util.ArrayList;
import java.util.ListIterator;
```

```
public class DvdTitleExpression extends DvdAbstractExpression {
   public String interpret(DvdInterpreterContext
                           dvdInterpreterContext) {
       ArrayList titles = dvdInterpreterContext.getAllTitles();
       ListIterator titlesIterator = titles.listIterator();
       StringBuffer titleBuffer = new StringBuffer("");
       boolean first = true;
       while (titlesIterator.hasNext()) {
           if (!first) {
               titleBuffer.append(", ");
           } else {
               first = false;
           }
           titleBuffer.append((String)titlesIterator.next());
       }
       return titleBuffer.toString();
   }
}
```

Wyjście na konsoli:

```
interpreting show actor: Query Result: Ethan Hawke, Denzel Washington
interpreting show actor for title : Query Result: Ethan Hawke, Denzel Washington
interpreting show title: Query Result: Caddy Shack, Training Day, Hamlet
interpreting show title for actor : Query Result: Hamlet, Training Day, Caddy Shack
```

11. Memento

Przeznaczenie (cytat z książki GoF):
Nie naruszając kapsułkowania zapamiętuje i udostepnia na zewnątrz stan wewnętrzny obiektu, tak że obiekt może być poźniej przywrócony do zapamiętanego stanu.
Przeznaczenie (cytat z fluffycat):
Jeden obiekt przechowuje stan innego obiektu.

Opis: Przykład ilustruje sposób wycofania się z niepoprawnego wzbogacenia wcześniej zapamiętanego tytułu.

Problem: Jak

Rozwiązanie: Przez

Klasa główna:

_DP_B_GoF_Memento

```
package pk.dydakt.to.dp.b.gof.memento;

import java.util.ArrayList;

public class _DP_B_GoF_Memento {

   public static void main(String[] args) {
       DvdDetails.DvdMemento dvdMementoCaretaker;
       //the Caretaker

       ArrayList<String> stars = new ArrayList<String>();
       stars.add(new String("Guy Pearce"));
       DvdDetails dvdDetails =
         new DvdDetails("Memento", stars, '1');
       dvdMementoCaretaker = dvdDetails.createDvdMemento();
       System.out.println("as first instantiated");
       System.out.println(dvdDetails.formatDvdDetails());

       System.out.println("");
       dvdDetails.addStar("edskdzkvdfb");
       //oops - Cappuccino on the keyboard!
       System.out.println("after star added incorrectly");
       System.out.println(dvdDetails.formatDvdDetails());

       System.out.println("");
       System.out.println("the memento");
       System.out.println(dvdMementoCaretaker.showMemento());
       //show the memento

       System.out.println("");
       dvdDetails.setDvdMemento(dvdMementoCaretaker);
       //back off changes
       System.out.println(
         "after DvdMemento state is restored to DvdDetails");
       System.out.println(dvdDetails.formatDvdDetails());
   }
}
```

Klasa wzorca:

DvdDetails

```
package pk.dydakt.to.dp.b.gof.memento;

import java.util.ArrayList;
import java.util.ListIterator;

//the originator
public class DvdDetails {
   private String titleName;
   private ArrayList<String> stars;
   private char encodingRegion;

   public DvdDetails(String titleName,
                     ArrayList<String> stars,
                     char encodingRegion) {
       this.setTitleName(titleName);
       this.setStars(stars);
       this.setEncodingRegion(encodingRegion);
   }

   private void setTitleName(String titleNameIn) {
       this.titleName = titleNameIn;
   }
   private String getTitleName() {
       return this.titleName;
   }

   private void setStars(ArrayList<String> starsIn) {
       this.stars = starsIn;
   }
   public void addStar(String starIn) {
       stars.add(starIn);
   }
   private ArrayList<String> getStars() {
       return this.stars;
   }
   private static String getStarsString(ArrayList starsIn) {
       int count = 0;
       StringBuffer sb = new StringBuffer();
       ListIterator starsIterator = starsIn.listIterator();
       while (starsIterator.hasNext()) {
           if (count++ > 0) {sb.append(", ");}
           sb.append((String) starsIterator.next());
       }
       return sb.toString();
   }

   private void setEncodingRegion(char encodingRegionIn) {
      this.encodingRegion = encodingRegionIn;
   }
   private char getEncodingRegion() {
       return this.encodingRegion;
   }

   public String formatDvdDetails() {
       return ("DVD: " + this.getTitleName() +
               ", starring: " + getStarsString(getStars()) +
               ", encoding region: " + this.getEncodingRegion());
   }

   //sets current state to what DvdMemento has
```

```
    public void setDvdMemento(DvdMemento dvdMementoIn) {
        dvdMementoIn.getState();
    }
    //save current state of DvdDetails in a DvdMemento
    public DvdMemento createDvdMemento() {
        DvdMemento mementoToReturn = new DvdMemento();
        mementoToReturn.setState();
        return mementoToReturn;
    }

    //an inner class for the memento
    class DvdMemento {
        private String mementoTitleName;
        private ArrayList<String> mementoStars;
        private char mementoEncodingRegion;

        //sets DvdMementoData to DvdDetails
        public void setState() {
             //Because String are immutable we can just set
             //  the DvdMemento Strings to = the DvdDetail Strings.
             mementoTitleName = getTitleName();
             mementoEncodingRegion = getEncodingRegion();
             //However, ArrayLists are not immutable,
             //  so we need to instantiate a new ArrayList.
             mementoStars = new ArrayList<String>(getStars());
        }
        //resets DvdDetails to DvdMementoData
        public void getState() {
             setTitleName(mementoTitleName);
             setStars(mementoStars);
             setEncodingRegion(mementoEncodingRegion);
        }

        //only useful for testing
        public String showMemento() {
             return ("DVD: " + mementoTitleName +
                     ", starring: " + getStarsString(mementoStars) +
                     ", encoding region: " + mementoEncodingRegion);
        }
    }
}
```

Wyjście na konsoli:

```
as first instantiated
DVD: Memento, starring: Guy Pearce, encoding region: 1

after star added incorrectly
DVD: Memento, starring: Guy Pearce, edskdzkvdfb, encoding region: 1

the memento
DVD: Memento, starring: Guy Pearce, encoding region: 1

after DvdMemento state is restored to DvdDetails
DVD: Memento, starring: Guy Pearce, encoding region: 1
```

Case-study w Java

Poniżej omówiono case-study przedstawione w książce GoF w wersji Java, na podstawie kodu źródłowego pochodzącego ze strony:

Potencjalne dziedziny techniczne do zastosowań wszystkich wzorców projektowych

1. **GUI**
2. Konwersja formatów zapisu
3. Bazy danych
4. **Edytor tekstu z wieloma dokumentami**
5. **Edytor graficzny np. do układów scalonych**
6. Środowisko programistyczne
7. **Gry komputerowe**
8. Komunikacja pomiędzy komputerami
9. Arkusz kalkulacyjny
10. Szkolenia i egzaminy on-line
11. System składania zamówień on-line
12. Sklep internetowy
13. Obsługa konta bankowego on-line

Potencjalne dziedziny życia codziennego do zastosowań wszystkich wzorców projektowych

14. Restauracja
15. Produkcja nagrań na CD
16. Handel
17. Biblioteka
18. Lecznictwo
19. Uprawa roślin
20. Hodowla zwierząt
21. Turystyka górska
22. Komunikacja np. lotnicza
23. Zarządzanie ludźmi
24. Obsługa samochodu
25. Sport
26. Narciarstwo
27. Zakupy
28. Rozliczenia uczestników wyjazdu
29. Planowanie wakacji

Warianty realizacji wzorców projektowych

Warianty:

- z klasami abstrakcyjnymi
- z interfejsami
- z wewnętrznymi klasami anonimowymi
- oparte na szablonach (typach uogólnionych)