# J2EE – wzorce projektowe

Alicja Truszkowska

# Motywacja

- Znaczące miejsce J2EE pośród systemów biznesowych
- Bogactwo narzędzi i technologii
- Dobre praktyki projektowania aplikacji

*“Każde, nawet najdoskonalsze narzędzia mogą zostać użyte nieprawidłowo lub nieefektywnie.”*

W. Crawford, J. Kaplan
*J2EE. Stosowanie wzorców projektowych*

# Plan prezentacji

1) Czym są wzorce projektowe
2) Zalety stosowania wzorców projektowych
3) Wprowadzenie do technologii J2EE
4) Przykłady wzorców projektowych
5) Literatura

# Definicja wzorca projektowego

*“Wzorce projektowe stanowią powtarzające się rozwiązania powtarzających się problemów. [...] Wzorzec dostarcza zbioru specyficznych interakcji, które mogą być zastosowane do ogólnych obiektów w celu rozwiązania znanego problemu.”*

W. Crawford, J. Kaplan
*J2EE. Stosowanie wzorców projektowych*

# Geneza wzorców projektowych

- Architektura

  *Chistropher Alexander*
  *A Pattern Language: Towns, Buildings, Construction*

- Gang of Four

  *E. Gamma, R. Helm, R. Johnson, J. Vlissides*
  *Design Patterns: Elements of Reusable Object Oriented Software*

# Anatomia wzorca projektowego

Najprostsze wzorce projektowe można wyrazić w jednym, czy dwóch zdaniach, np.

*wykorzystanie bazy danych przez witrynę internetową*

# Anatomia wzorca projektowego
## planowanie miasta

Nazwa: zabudowa blokowa (układ siatki)

Siły:

- trudność poruszania się w wielkim mieście, w którym drogi nie biegną równolegle do siebie
- łatwość budowania budynków na planie prostokąta

Uczestnicy: budynki, ulice, chodniki, piesi,  samochody

Rozwiązanie: szczegóły rozmieszczenia

# Anatomia wzorca projektowego
## planowanie miasta

Uogólnienie wzorca:

| | **miasto** | **biuro** | **restauracja** |
|---|---|---|---|
| **komponenty** | *budynki* | *pokoje* | *stoliki* |
| **drogi** | *ulice*<br>*chodniki* | *korytarze* | *przejścia* |
| **podróżni** | *ludzie*<br>*samochody* | *pracownicy* | *kelnerzy* |

# Przykład zrealizowanego wzorca

np. Nowy Jork, Waszyngton, Filadelfia

ale np. Boston, Warszawa już nie

www.bigdig.com

# Najważniejsze cechy wzorców projektowych

- powstają w wyniku prób i błedów
- zapisywane w postaci struktur
- zapobiegają ponownemu wynajdowaniu koła
- istnieją na różnych poziomach abstrakcji
- podlegają ciągłym udoskonaleniom
- można je wykorzystywać wielokrotnie
- popularyzują projekty oraz najlepsze techniki programowania
- można je ze sobą łaczyć, aby rozwiązać większy problem

# J2EE

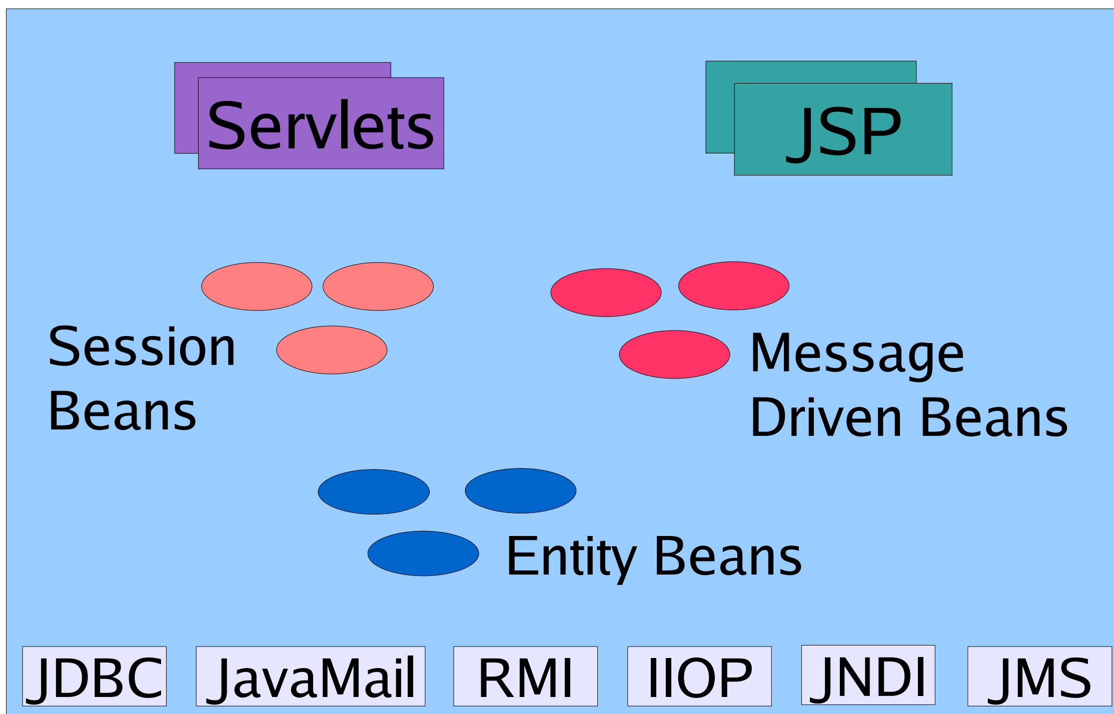
Aplikacja klienta

HTML

XML

Applet

Warstwa prezentacji po stronie klienta

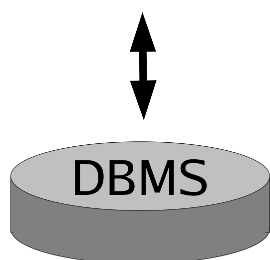


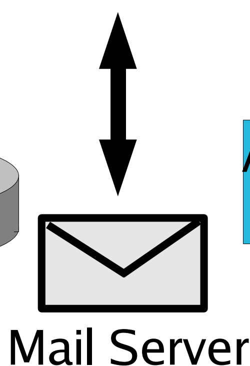

Warstwa prezentacji po stronie serwera

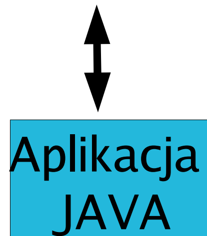
Warstwa logiki biznesowej

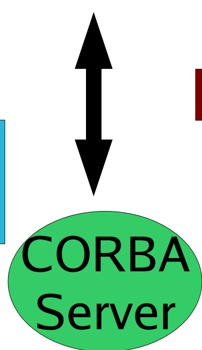
Warstwa modelu danych

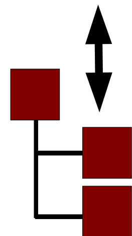
Warstwa integracji

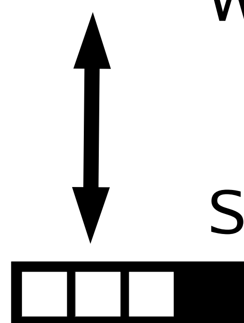




Systemy zewnętrzne

# Kryteria jakości aplikacji

- rozszerzalność – *łatwość wprowadzania zmian*
  - *usuwanie powiązań*
  - *centralizacja*
  - *ponowne użycie*
- skalowalność – *zachowanie wydajności przy zwiększonym obciążeniu*
- niezawodność
- terminowość

# Warstwa prezentacji

- stanowi interfejs użytkownika
- określa zestaw funkcji udostępnianych użytkownikowi
- nacisk na rozszerzalność
  (od tego zależy łatwość dodawania kolejnych funkcjonalności)

```
public class MagicServlet extends HttpServlet {
	private DirContext peopleContext;

	public void init (ServletConfig config) throws Servlet Exception {
		super.init(config);
		Properties env = new Properties();
		env.put (Context.INITIAL_CTX_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
		env.put (Context.PROVIDER_URL, "ldap://localhost/o=jndiTest");
		try {
			DirContext initalContext = new InitialDirContext(env);
			peopleContext = (DirContext) initalContext.lookup("ou=people");
		catch (NamingException ne) {
			ne.printStackTrace();
			throw new ServletException("Błąd podczas inicjacji LDAP", ne);
		}
	}

	public void destroy() {
		try {
		peopleContext.close();
		} catch(NamingException ne) {
			ne.printStackTrace();
		}
	}
```

```
    protected void doGet(HttServletrequest request, HttpServletResponse response)
    throws Servlet Exception, IOException {

        response.setContentType("text/html");
        java.io.PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<table>");

        try {
        NamingEnumeration people = peopleContext.list("");
        while(people.hasMore()) {
            NameClassPair personName = (NameClassPair) people.next();
                Attributes personAttrs = peopleContext.getAttrs (personName.getName());
                Attribute name = personAttrs.get("name");
                Attribite phone = peronAttrs.get("tel");
                out.println("<tr><td>" + name.get() + "</td><td>" + phone.get() +"</td></tr>");
            }
        } catch (Exception ex) {
            out.println ("Bład podczas pobierania danych");
        }

        out.println("</table>");
        out.println("</body>");
        out.println("</html>");
    }
} //MagicServlet
```

# Model

```
public class Person {
      private String name;
      private String tel;

      . . .

      public String getName() {return name};
      public String getTel() {return tel};

      public void setName(String n) {name = n};
      public void setTel(String t) {tel = t);
      }
}

public interface PersonCommand {
      public void initialize (HttpSession session) throws NamingException;
      public void runCommand() throws NamingException;
      public List getPeople(); // zwraca listę obiektów Person

public class LdapPersonCommand() implements PersonCommand {
      . . .
}
```

* wzorce: lokalizator usługi, obiekt wartości

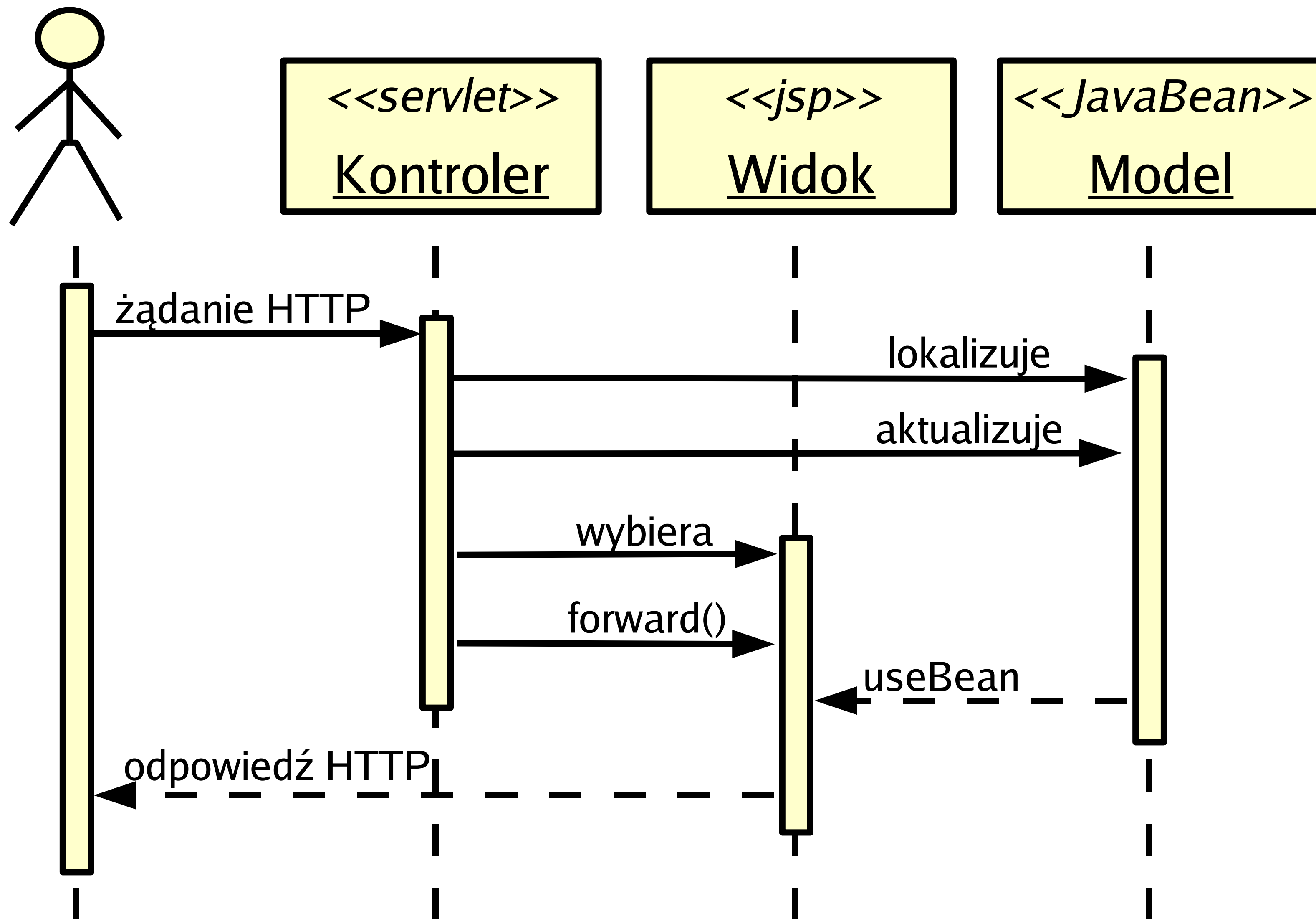
# Widok

```
/*PersonView.jsp*/

<%@page contentType=”text/html”%>
<%@taglib uri=”/jstl/core” prefix=”c”%>

<html>
<head>
    <title>Książka adresowa</tilte>
<head>

<body>
    <jsp:useBean id=personCommand” scope=”request”
                class=”web.model.PersonCommand” />

    <table>
        <c:forEach var=”person” items=”$(personCommand.people)”>

            <tr><td><c:out value=”$(person.getName())”/></td>
               <td><c:out value=”$(person.getTel())”/></td>
            </tr>

        </c:forEach>
    </table>

</body>
</html>
```

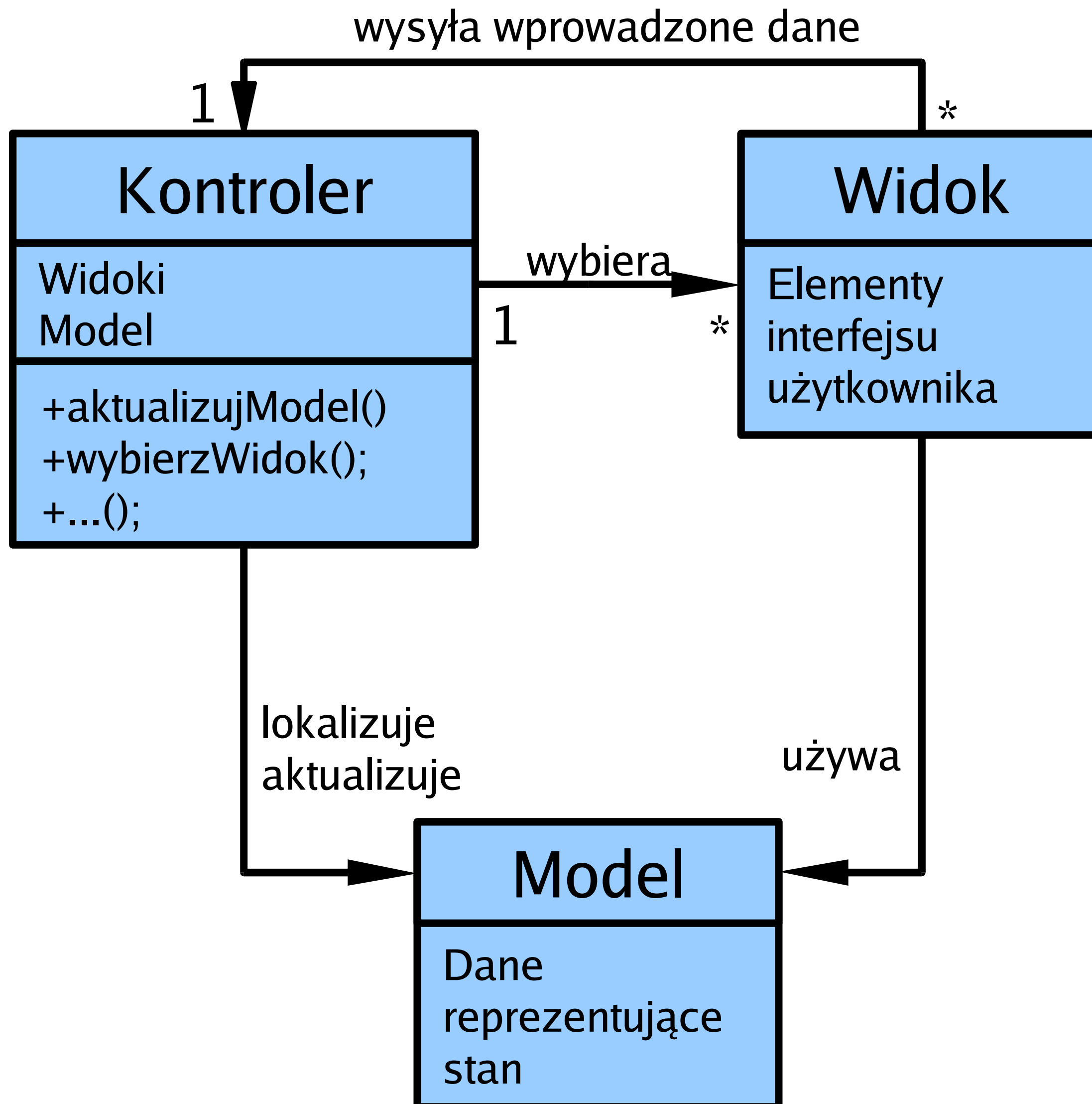
# Kontroler

```
public class PersonServlet extends HttpServlet {
    protected void doGet(HttServlet request request, HttpServletResponse
response)
    throws Servlet Exception, IOException {
        try {
        PersonCommand personCommand = new LdapPersonCommand();
        personCommand.initialize(request.getSession());
        personCommand.runCommand();

        request.setAttribute("personCommand", personCommand);
        } catch (NamingException ne) {
            throw new ServletException("Błąd wykonania polecenia", ne);
        }

        RequestDispatcher dispatch =
            getServletContext().getREquestDispatcher("/PersonView.jsp");
        dispatch.forward(request, response);
    }
}
```

# Model – Widok - Kontroler



# Model – Widok - Kontroler




- dostarcza architektury dla całej warstwy prezentacji
- umożliwia separację stanu, prezentacji i zachowania
- każdy z elementów mogą opisywać kolejne wzorce projektowe

# Wzorce kontrolera

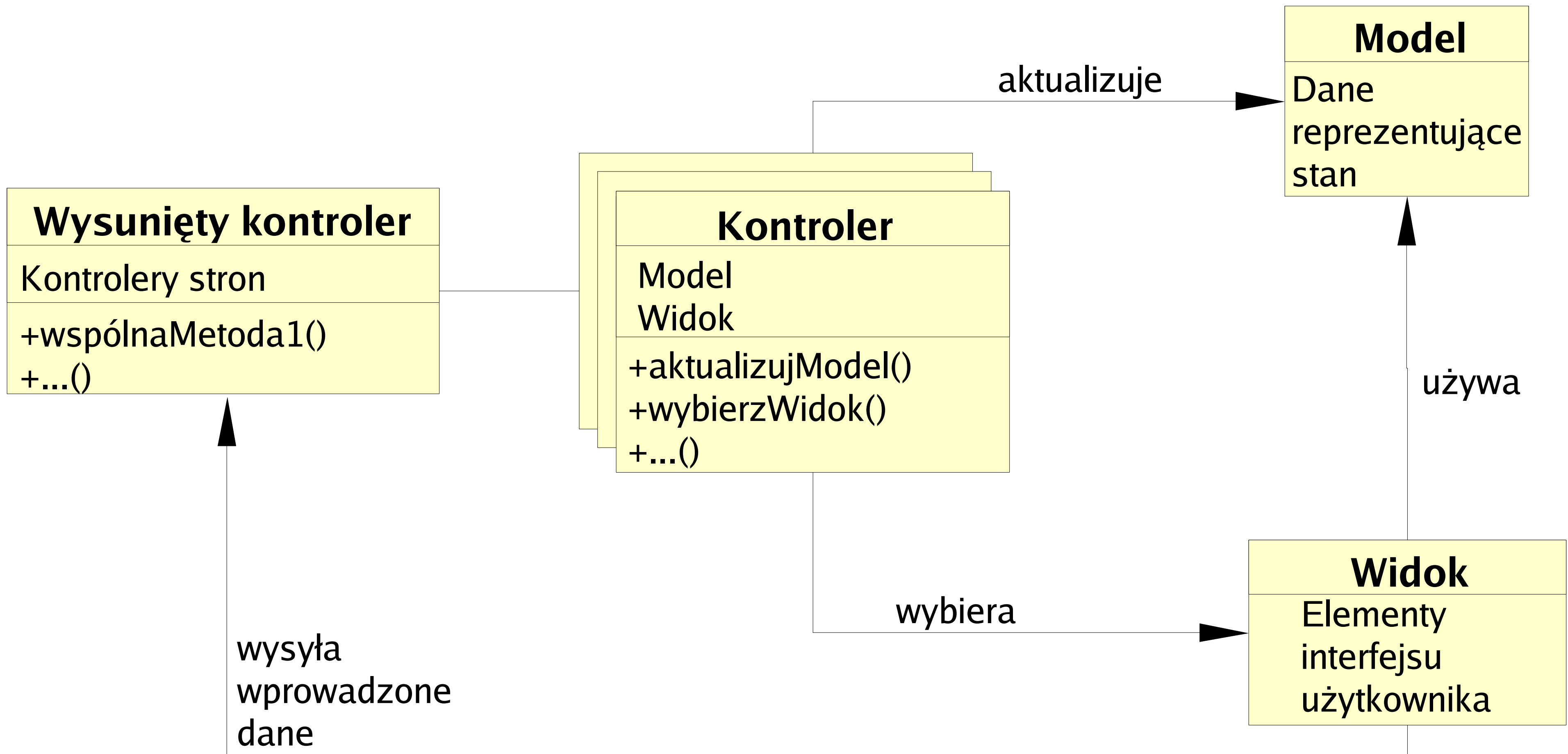
Ile powinno być kontrolerów?

- jeden kontroler obsługujący wszystkie żądania
- osobny kontroler do każdego żądania

Rozwiązanie:

wzorzec wysuniętego kontrolera

# Wzorzec wysuniętego kontrolera



# Różnice w obsłudze żądań

Problem: część akcji wykonywana tylko dla niektórych żądań
(np. zapis do dziennika, sprawdzenie sposobu kodowania)
wysunięty kontroler nie wystarcza

Rozwiązanie 1: warunki wykonania kodu dla każdego żądania
są opisane w wysuniętym kontrolerze

Rozwiązanie 2: utworzenie kilku wysuniętych kontrolerów, każdy
z inną sekwencją opcjonalnych fragmentów kodu

Rozwiązanie 3: opcjonalny kod zapisany w kontrolerach stron

Rozwiązanie 4: różne łańcuchy filtrów dostosowane do każdego
z żądań, zbudowane za pomocą dekoratorów

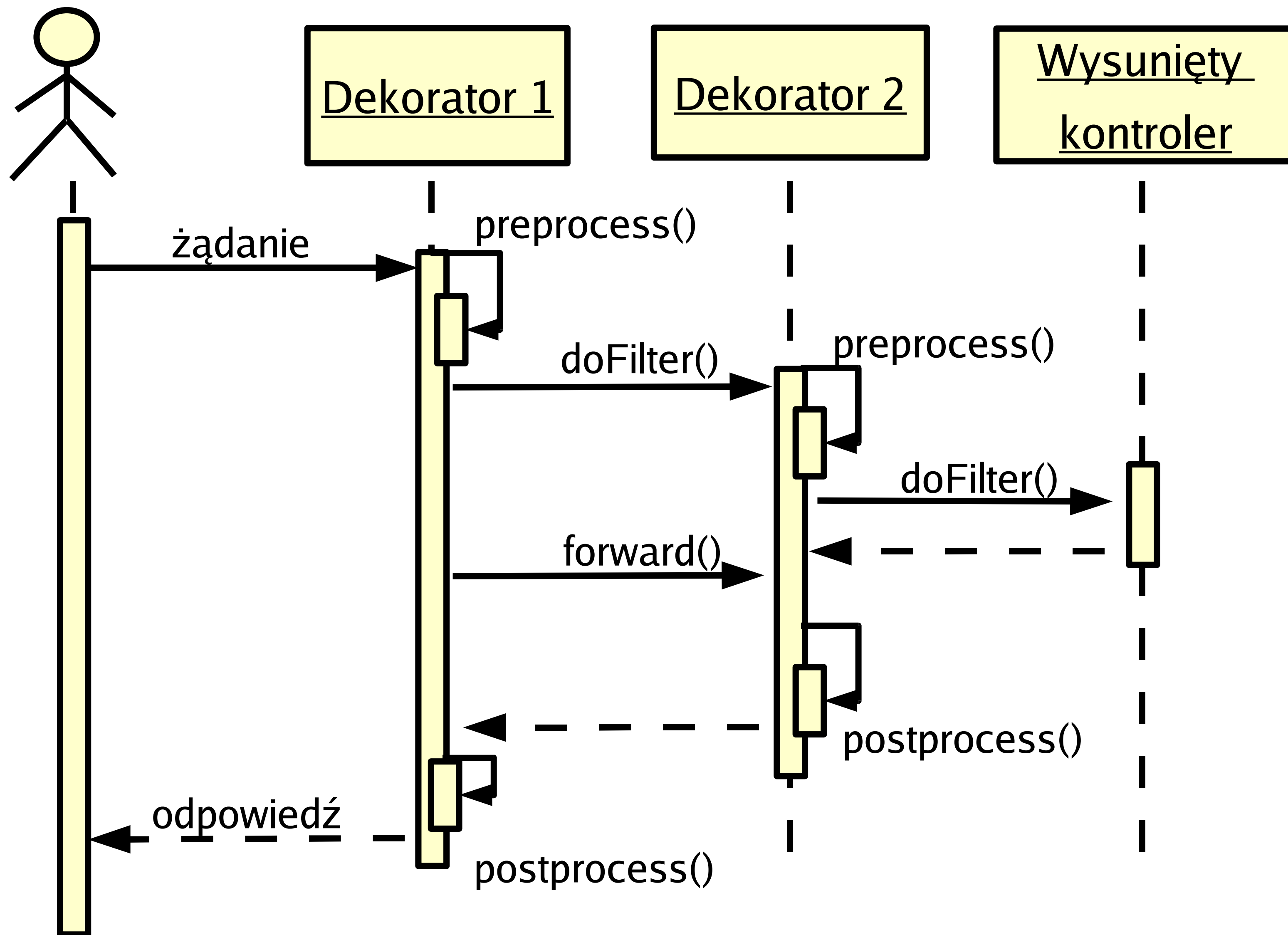
# Wzorzec filtra dekorującego

Dekorator – klasa obudowująca, zawierająca pojedynczy obiekt podrzędny i mająca taki sam interfejs jak ten obiekt

Wywołanie metody opakowanej klasy:

1. wykonanie kodu metody z klasy opakowującej
2. przekazanie sterowania do metody obiektu opakowanego
3. powrót do metody klasy opakowującej

# Wzorzec filtra dekorującego



# Usługa i wykonawca

Problem: ścisłe powiązanie stron i dostępnych akcji z kontrolerem

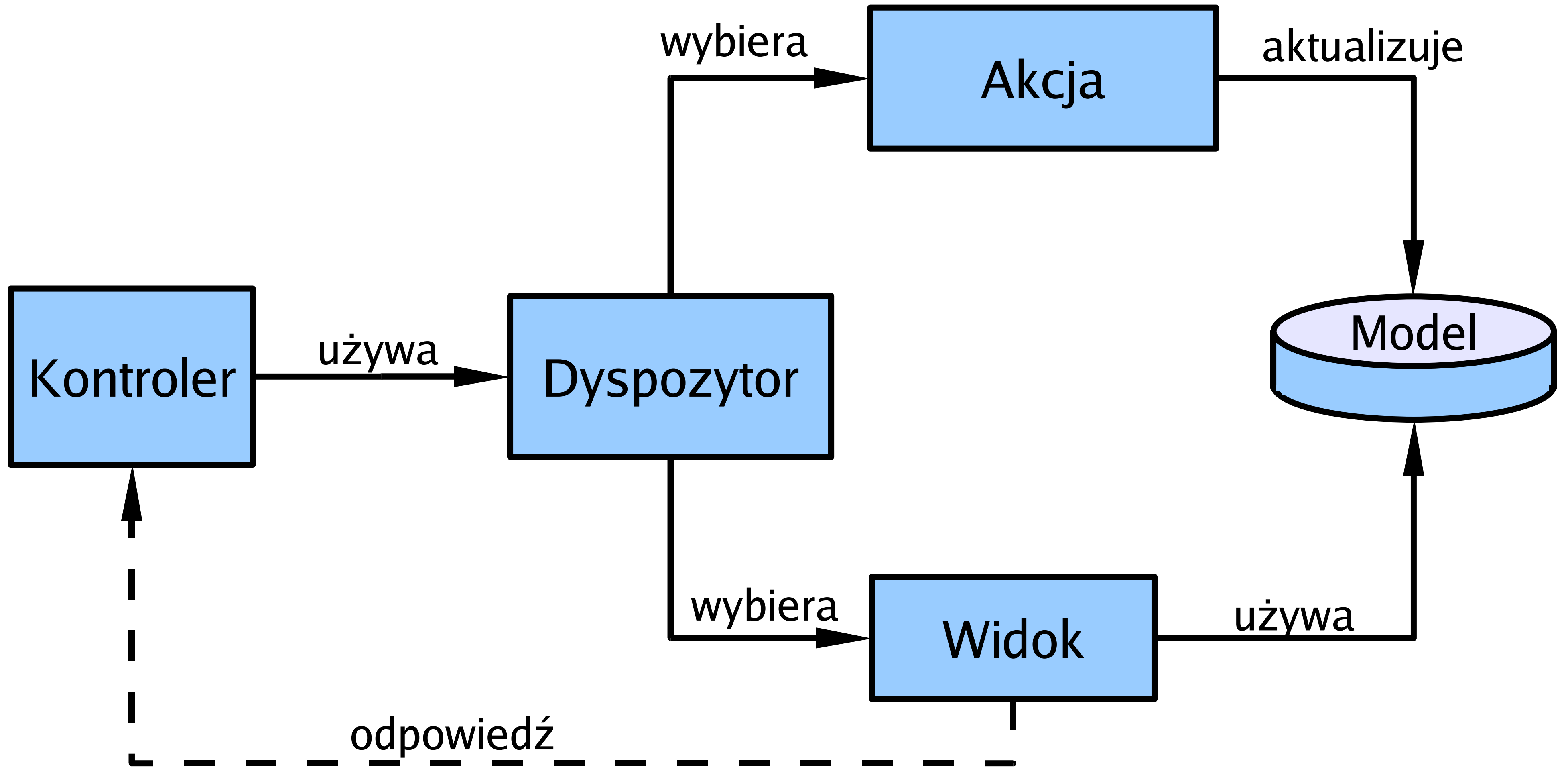
Rozwiązanie: strony i akcje jako oddzielne klasy

Możliwości:

- Elastyczne dodawanie i usuwanie stron
- Używanie kilka razy tej samej strony

Struts

# Usługa i wykonawca



# Pomocnik widoku

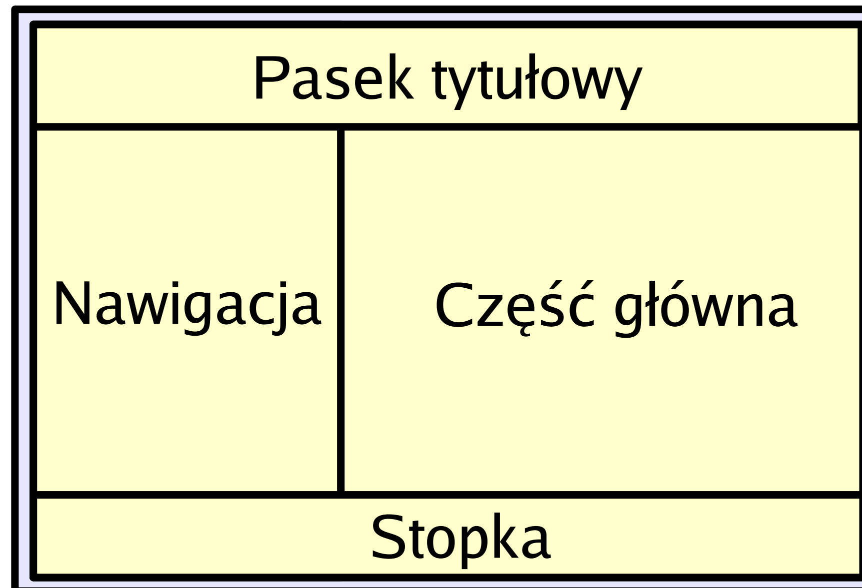
- oddzielenie prezentacji od formatowania
- unikanie rozbudowanych stron JSP
- unikanie powielania kodu
- definiowanie znaczników (Struts)

```
<%
    EmployeeDelegate empDelegate = new EmployeeDelegate();
    Iterator employees =
empDelegate.getEmployes(EmployeeDelegate.ALL_DEPARTMENTS);
%>

<table border ="1" >
    <tr>
        <th> Imi  </th>
        <th> Nazwisko </th>
        <th> Stanowisko </th>
    </tr>

<%
    while (employees.hasNext() )
    {
        EmployeeVO employee = (EmployeeVO) employees.next();
%>
    <tr>
        <td> <%= employee.getFirstName().toUpperCase() %> </td>
        <td> <%= employee.getLastName().toUpperCase() %> </td>
        <td> <%= employee.getDesignation() %> </td>
    </tr>

<%
        }
    }
%>
</table>
```

```
<patterns:employeeListGet />

<table border ="1" >
    <tr>
        <th> Imi  </th>
        <th> Nazwisko </th>
        <th> Stanowisko </th>
    </tr>

    <patterns:employeeList id="employeeList">

    <tr>
        <td> <patterns:employee attr="FirstName" case="Upper"> </td>
        <td> <patterns:employee attr="LastName" case="Upper"> </td>
        <td> <patterns:employee attr="Designation">  </td>
    </tr>

    </patterns:employeeList>

</table>
```

# Widok złożony

Rozkład widoku na ogólny szkielet, który następnie zapełniamy niezależnymi elementami.

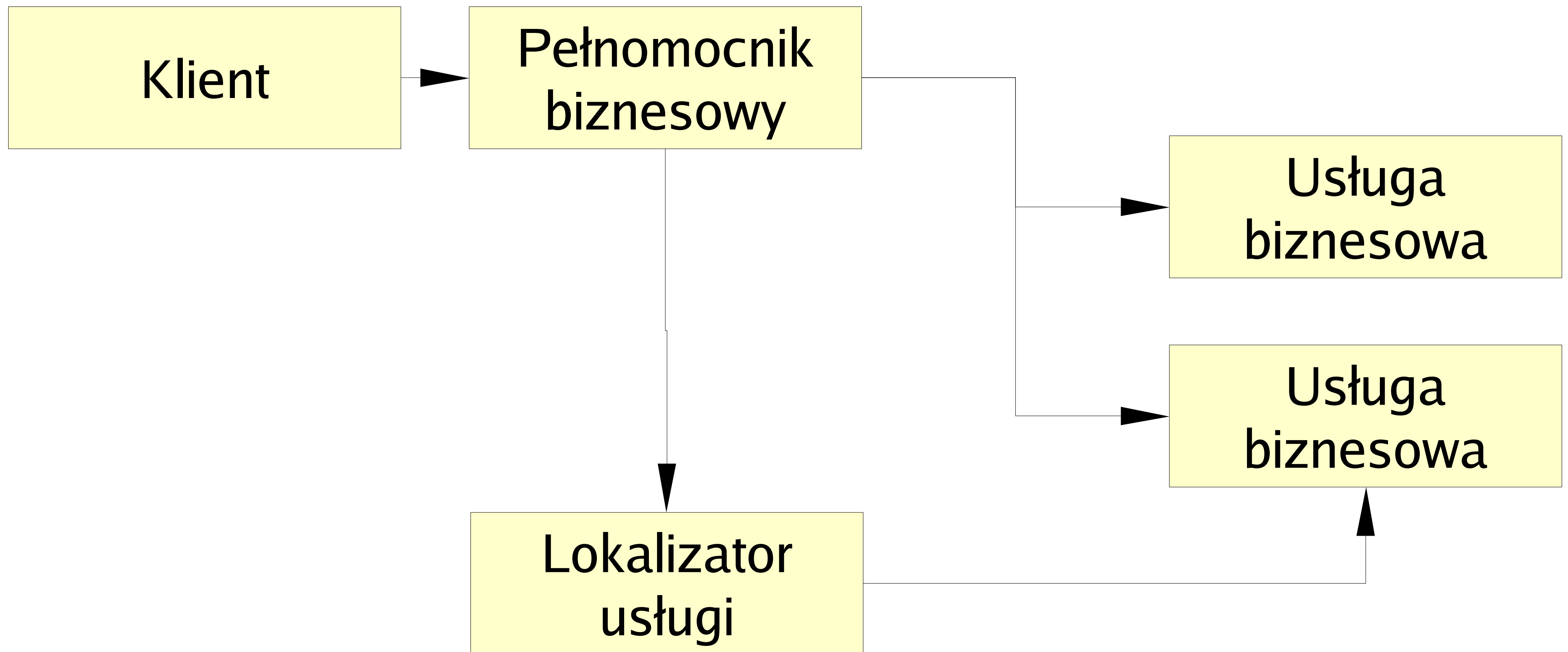



Przykład: biblioteka Tiles

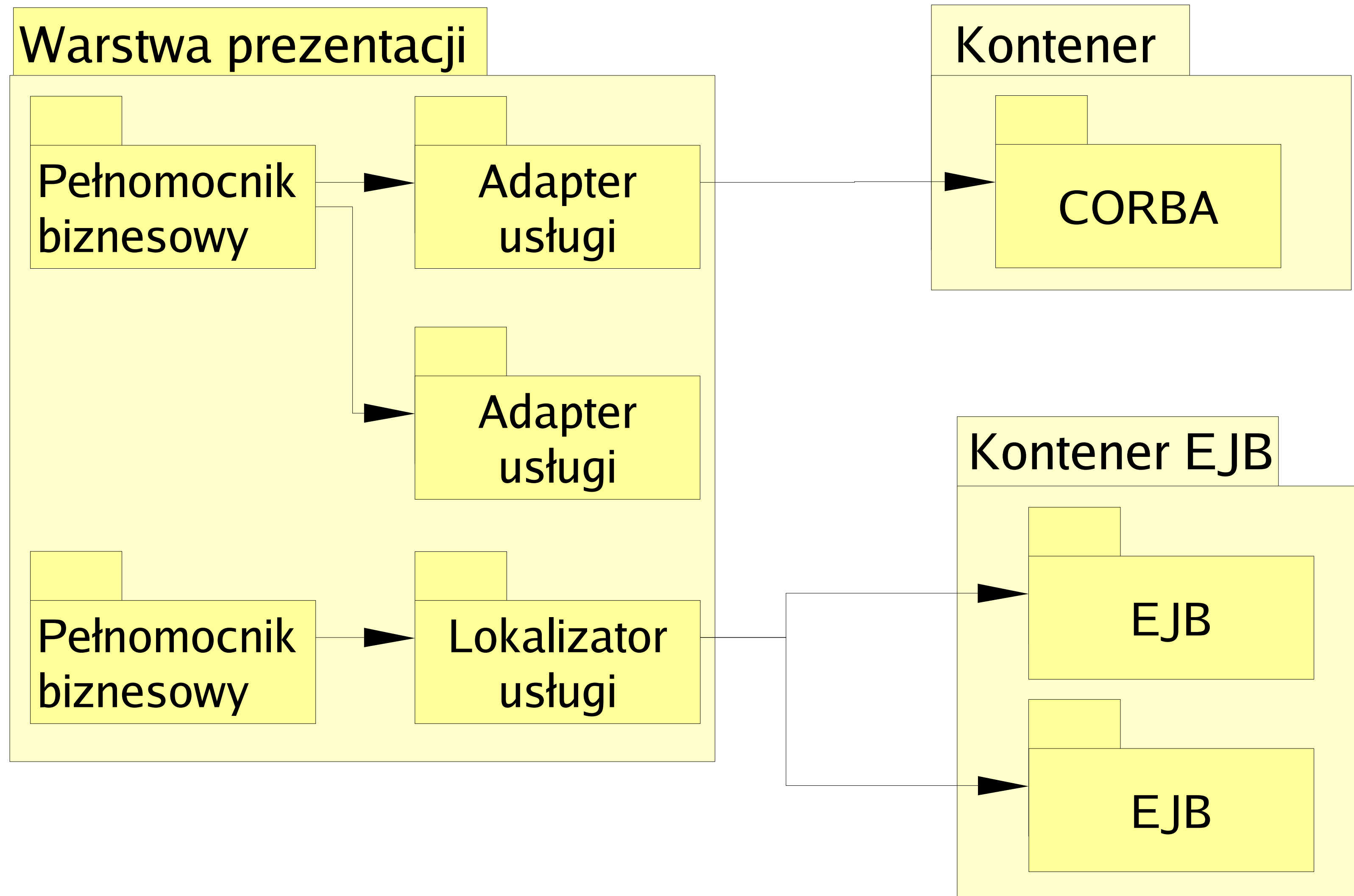
# Warstwa biznesowa

- Pełnomocnik biznesowy
- Fabryka pełnomocników biznesowych

# Warstwa biznesowa



# Warstwa biznesowa



# Warstwa bazy danych

Obiekty dostępu do danych(DAO)

Pomysł: jeden obiekt = jeden wiersz tabeli

np.

| Uczeń |
| --- |
| imię<br>nazwisko |

| Ocena |
| --- |
| uczeń<br>przedmiot<br>wartość |

Pytanie: Co się dzieje, gdy zapytamy o wszystkie oceny danego ucznia?

# Warstwa bazy danych

Rozwiązanie: Byt złożony

| Uczeń |
| --- |
| imię<br>nazwisko<br>kolekcja ocen |

Związane wzorce:

- isDirty
- odroczone ładowanie
- Fabryka DAO

# Literatura


- W. Crawford, J. Kaplan *J2EE. Stosowanie wzorców projektowych*
- D. Alur, J Crupi, D. Malks – *Core J2EE. Wzorce projektowe*
- Joshua Kerievsky - *Refaktoryzacja do wzorców projektowych*
- E. Gamma, R. Helm, R. Johnson, J. Vlissides
  *Design Patterns: Elements of Reusable Object Oriented Software*
- M. Fowler – *Patterns of Enterprise Application Architecture*