

SOA - JAVA EE - warstwa prezentacyjna

JSP, JSF

JSP

Cykl życia strony JSP

- Strony JSP są przetwarzane jako servlety, więc dziedziczą po nich wiele cech
- Kiedy następuje odwołanie do strony JSP, jest ona tłumaczona na kod źródłowy Javy, do klasy będącej servletem. Następuje kompilacja.
- Cykl życia strony JSP jest identyczny jak cykl życia servleta
 - Gdy nie istnieje instancja servletu danej strony JSP, serwer
 - Ładuje klasę servletu JSP
 - Tworzy instancję tego servletu
 - Wywołuje metodę `jspInit()`
 - Wywoływana jest metoda `_jspService()`
 - Przed usunięciem strony JSP następuje wywołanie funkcji `jspDestroy()`

Elementy stron JSP

- Dyrektywy
 - `<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>`
- Elementy języka wyrażeń (expression language) `${}` i `#{}`
- Znaczniki `<jsp: .. />`
- Inne znaczniki
 - Z bibliotek znaczników
 - Własne znaczniki

JavaBeans w JSP

- `<jsp:useBean id="nazwaBeana" class="pełna.nazwa.klasy" scope="widocznosc" />`
- `<jsp:useBean id="nazwaBeana" class="pełna.nazwa.klasy" scope="widocznosc" >`
 `<jsp:setProperty ...>`
`</jsp:useBean>`
 - `<jsp:setProperty name="nazwaBeana" property="nazwaWlasciwosci" value="wartosc|wyrazenie"/>`
 - `<jsp:setProperty name="nazwaBeana" property="nazwaWlasciwosci" param="wartosc|wyrazenie"/>`
- Widocznosc: **application, session, request**

Język wyrażeń EL (expression language)

- Są dwa typy wyrażeń
 - `${wyrażenie}` – natychmiastowa ewaluacja
 - Tylko do odczytu
 - Obliczenie po pierwszym wyświetleniu strony
 - `#{wyrażenie}`
 - Do odczytu i zapisu
 - Obliczenia w dowolnym etapie cyklu życia strony
- Użycie
 - W statycznym tekście, w atrybucie znacznika
 - Wyrażenie jest automatycznie konwertowane w zależności od typu (int, String, ..)

Język wyrażen

- `${zmienna}`
 - Serwer będzie szukał zmiennej o podanej nazwie kolejno w kontekstach:
 - Strony
 - Żądania
 - Sesji
 - Aplikacji
- Dostęp do właściwości obiektów jest możliwy przez
 - Kropkę .
 - `${osoba.imie}`
 - `[]`
 - `#{osoba[imie]}`

Język wyrażeń

- $\{a.b\}$ lub $\{a[b]\}$
 - Jeżeli zmienna jest bean-em, zwróci właściwość b a.getB()
 - Jeżeli zmienna a jest mapą, wówczas zwróci obiekt pod kluczem b
 - Jeżeli zmienna jest listą, zwróci obiekt spod indeksu b
- Operatory
 - +, -, *, /, div, %, mod
 - and, &&, or, ||, not, !
 - ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le
 - empty

Język wyrażen

- Zmienne predefiniowane
 - pageContext
 - servletContext
 - session
 - request
 - response
 - param
 - paramValues
 - header
 - headerValues
 - Cookie
 - initParam

Biblioteki znaczników

- Użycie bibliotek znaczników:
 - Deklaracja
 - `<%@ taglib prefix="prefix" tagdir="/WEB-INF/tags/dir" %>`
 - `<%@ taglib prefix="prefix" uri="URI" %>`
 - Użycie
 - `<prefix:tag attr1="value" attr2="value" ... />`
- Dołączanie fragmentów stron
 - `<%@ include file="filename.jspf" %>`
 - `<jsp:include page="includePage.jspf" />`
- Przekazywanie żądań
 - `<jsp:forward page="/main.jsp" />`

Java Standard Tag Library JSTL

- Standardowa biblioteka znaczników JSP
- Core:
 - remove
 - set
 - choose
 - forEach
 - forTokens
 - if
 - import
 - redirect
 - url
 - catch
 - out
- I18n
 - setLocale
 - requestEncoding
 - bundle
 - message
 - setBundle
 - formatNumber
 - formatDate
 - parseDate
 - parseNumber
 - setTimeZone
 - timeZone
- Database
 - setDataQuery
 - query
 - transaction
 - update
- Functions
 - length
 - toUpperCase
 - substring
 - trim
 - replace
 - indexOf
 - split, join
 - escapeXML

JSF

Java Server Faces JSF

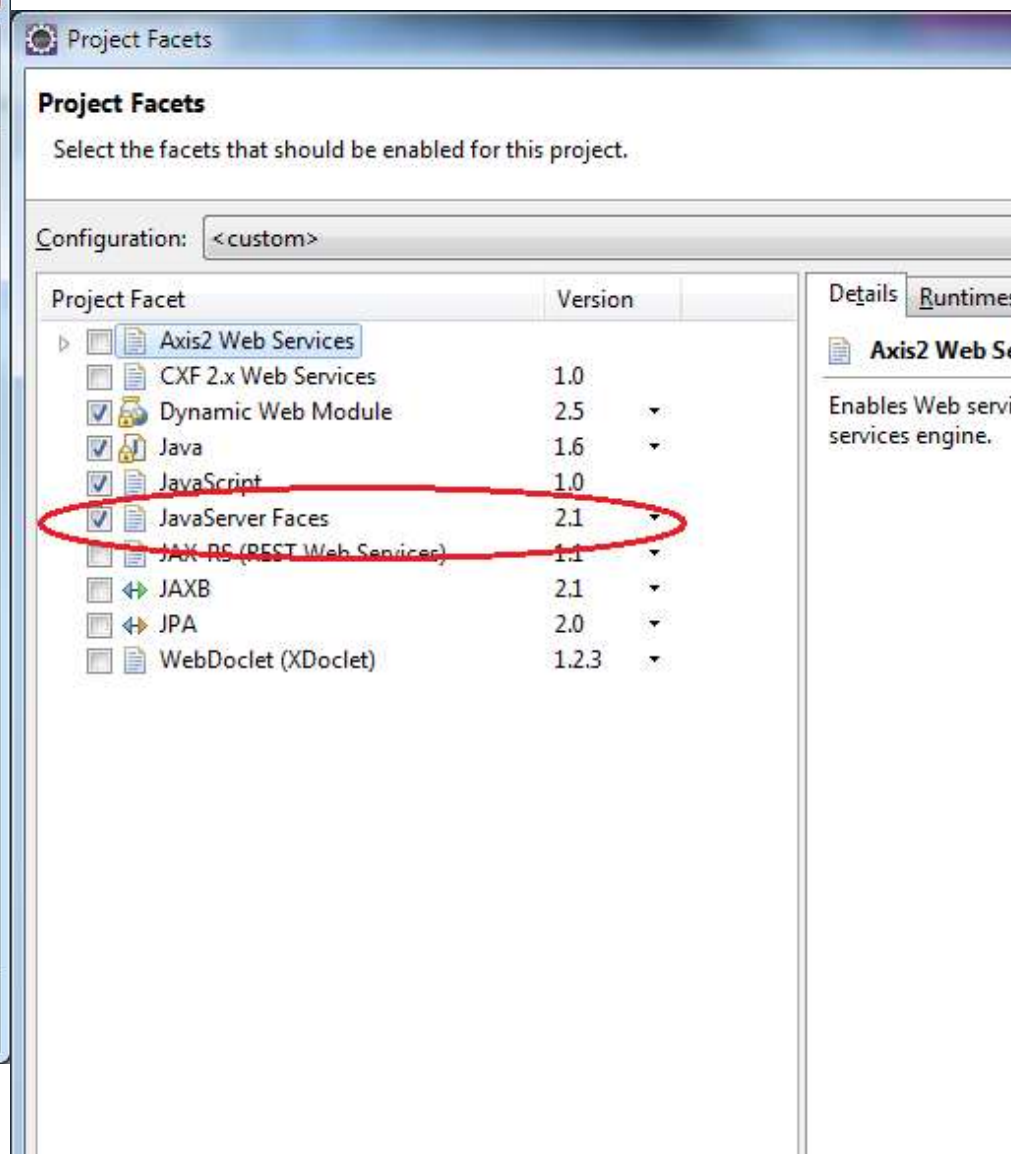
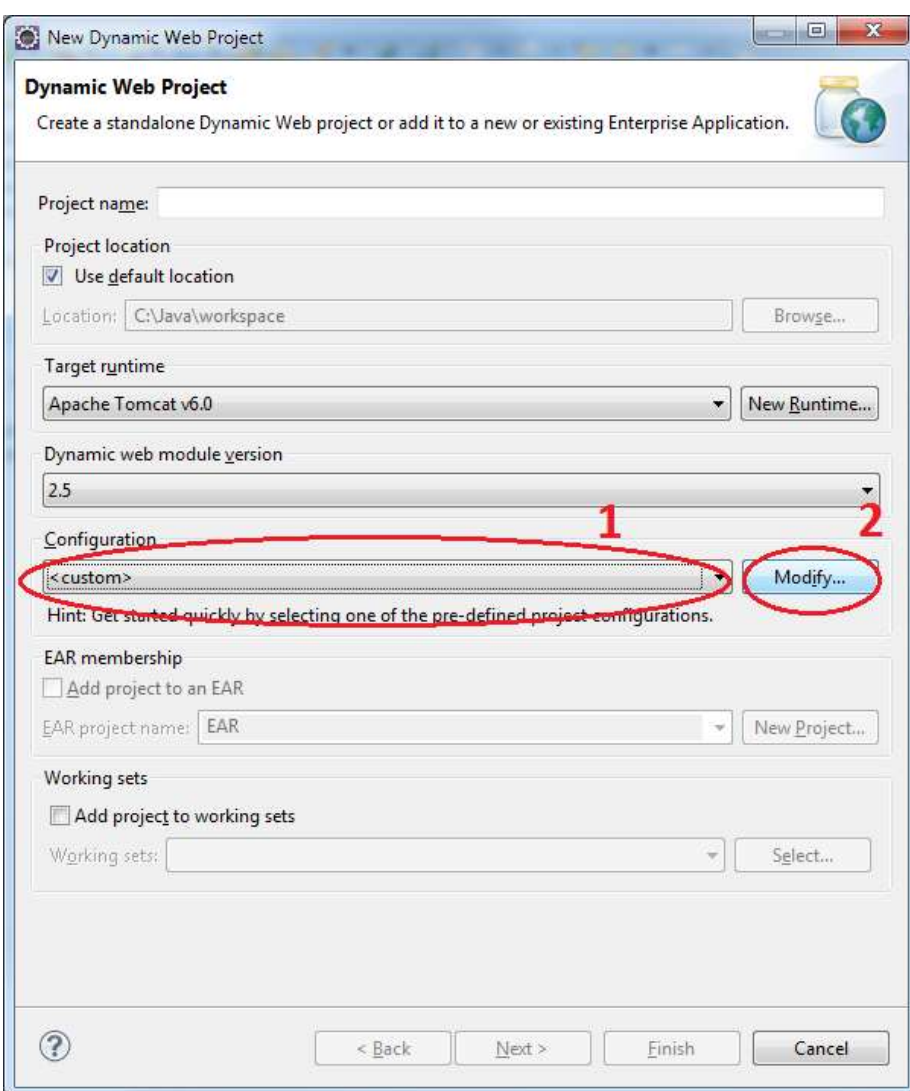
- Technologia, ułatwia i standaryzuje tworzenie interfejsu użytkownika w aplikacjach WWW na platformie Java EE
- Część specyfikacji Java EE (od J2EE 1.4)
- Oparta o stanowe, konfigurowalne komponenty interfejsu użytkownika pracujące po stronie serwera
- Koncepcja podobna do Web Forms w ASP.NET
- Główne elementy technologii JSF:
 - API do reprezentowania komponentów interfejsu użytkownika i zarządzania nimi
 - biblioteki znaczników dla JSP/Facelets

- Zakres funkcjonalny:
 - stanowy, komponentowy interfejs użytkownika
 - obsługa nawigacji między stronami
 - walidacja danych
 - wsparcie dla aplikacji wielojęzycznych
- JSF jako implementacja architektury MVC:
 - model komponentów do tworzenia stron-widoków
 - gotowy, konfigurowalny kontroler: FacesServlet
 - nie obejmuje modelu, współpracuje z różnymi modelami

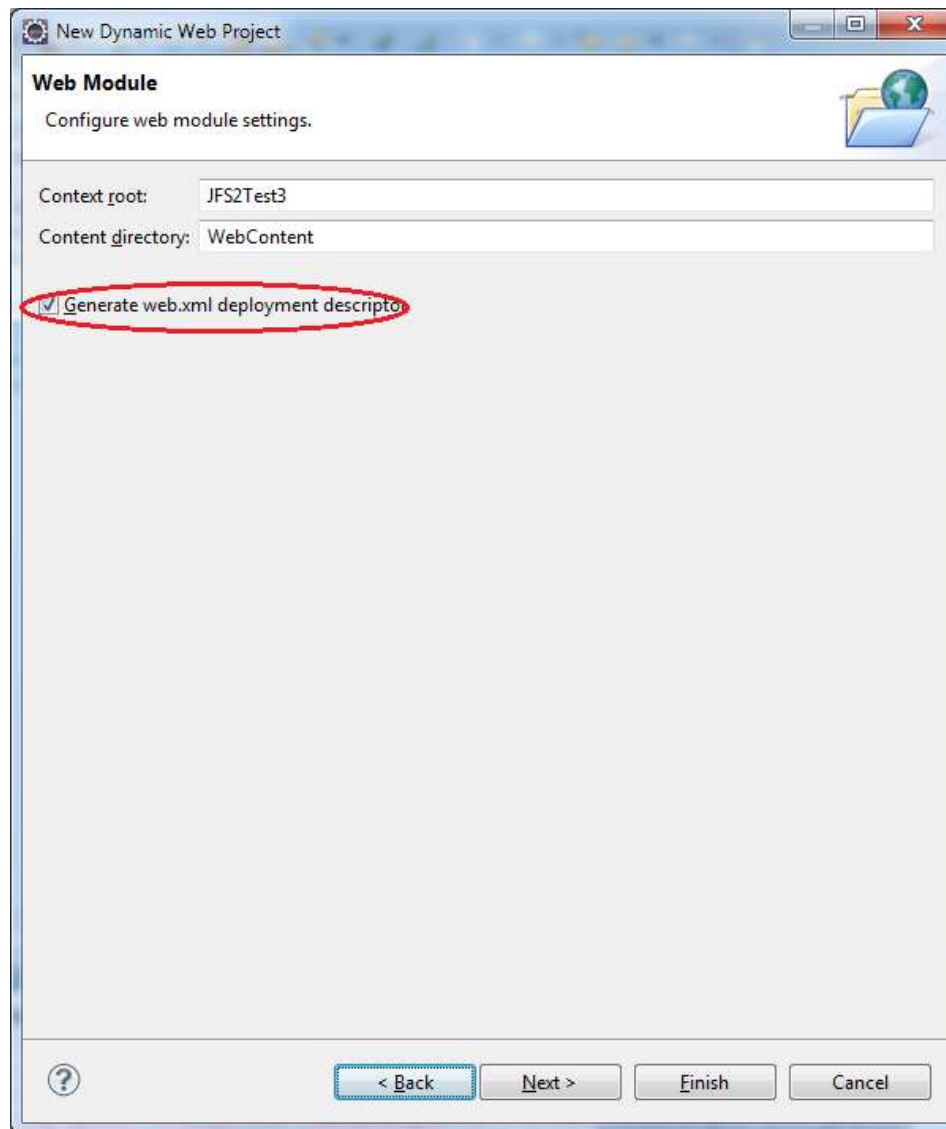
Tworzenie aplikacji JSF

- Konfiguracja serwletu kontrolera jako punktu wejścia do aplikacji
- Utworzenie stron-widoków z wykorzystaniem znaczników odwołujących się do komponentów JSF
- Zdefiniowanie nawigacji między stronami w pliku konfiguracyjnym aplikacji (od wersji 2.0 nie wymagane).
- Implementacja komponentów JavaBean reprezentujących właściwości i funkcje komponentów dla stron (tzw. Backing Beans)
- Zadeklarowanie komponentów JavaBean, których cyklem życia ma zarządzać JSF w pliku konfiguracyjnym jako tzw. managed beans (od wersji 2.0 nie wymagane).

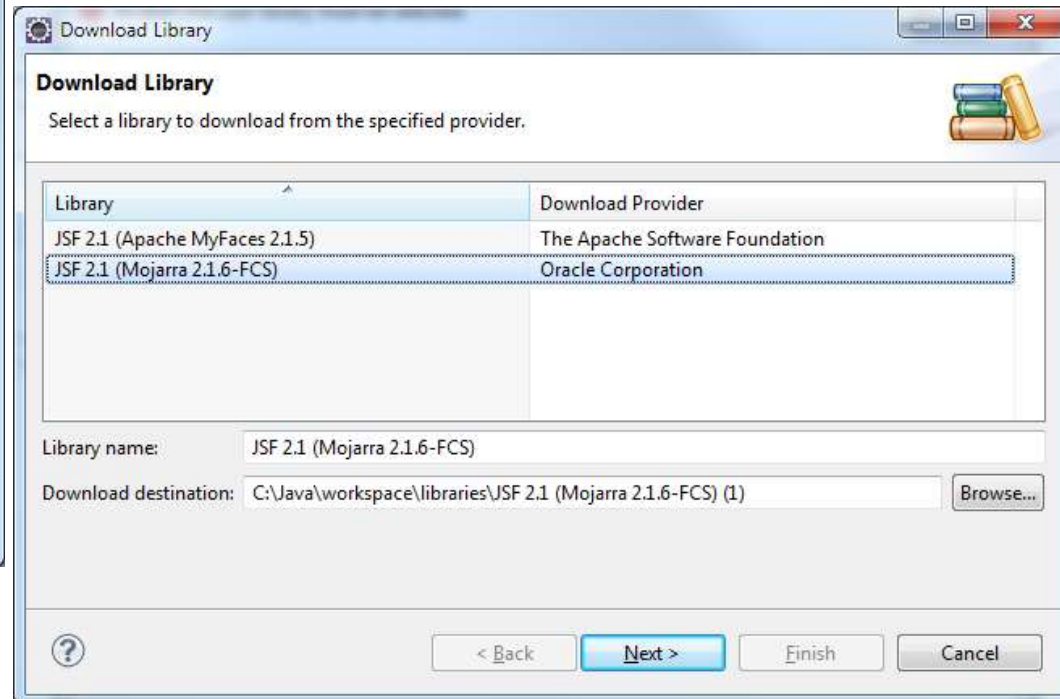
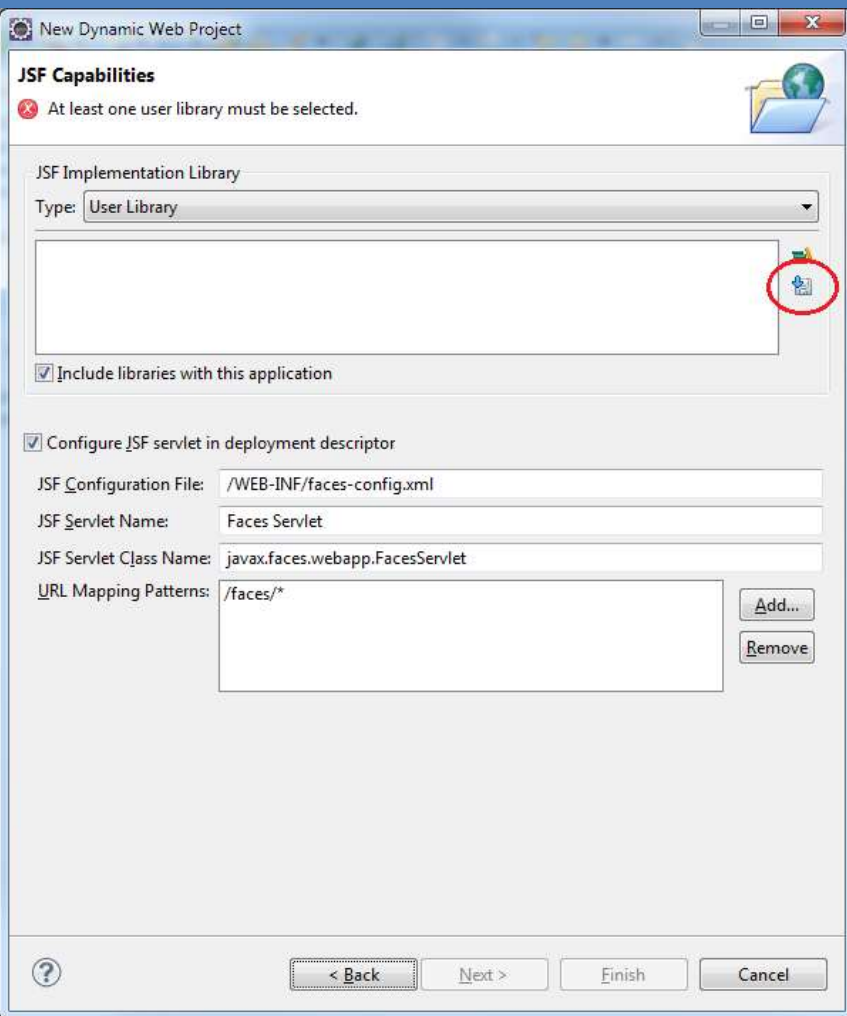
Projekt w Eclipse



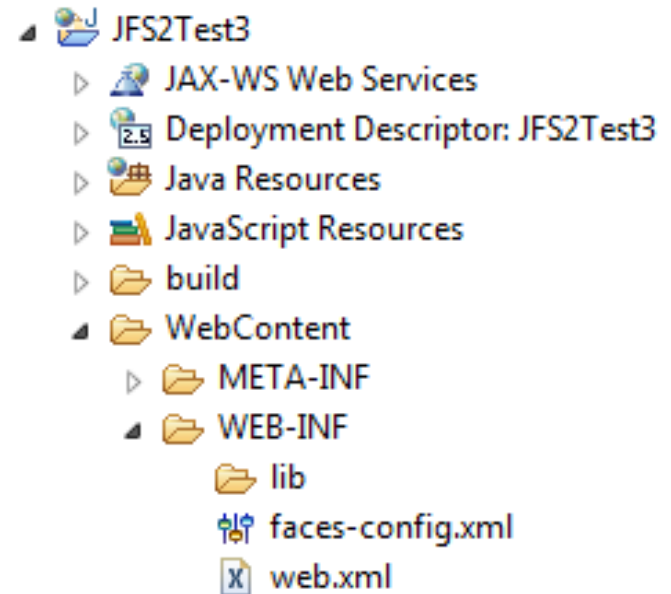
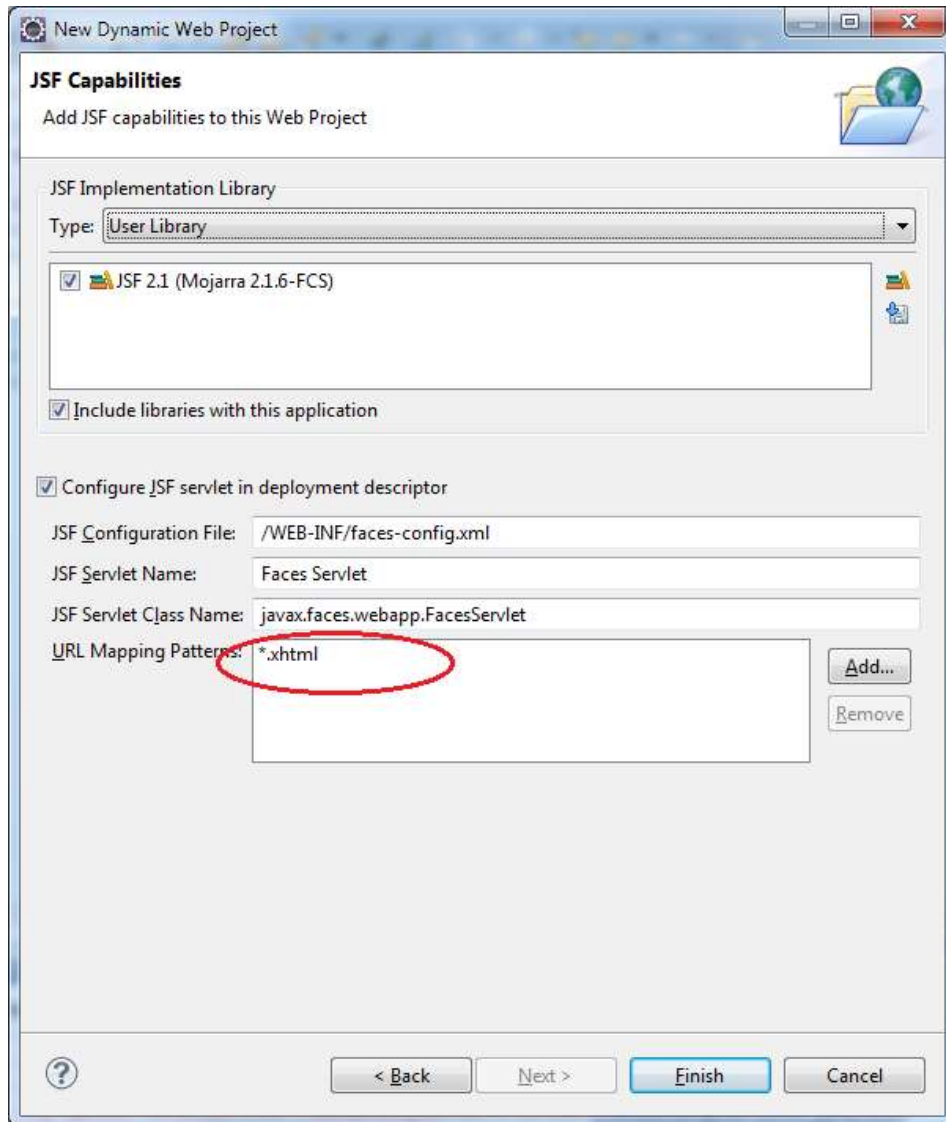
Projekt w Eclipse



Projekt w Eclipse



Projekt w Eclipse




Plik web.xml

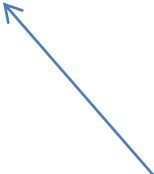
```
<web-app ... version="2.5">
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class> javax.faces.webapp.FacesServlet      </servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
  </servlet-mapping>

  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
```

Zamiast *.xhtml często
wpisywane są również *.jsf lub
/faces/



Nowość w JSF 2.
na stronie [www](#) zamieszczane są dodatkowe
informacje podczas pracy aplikacji



faces-config.xml


```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
http://java.sun.com/xml/ns/javaee/web-facesconfig_2_1.xsd"  
  version="2.1">
```

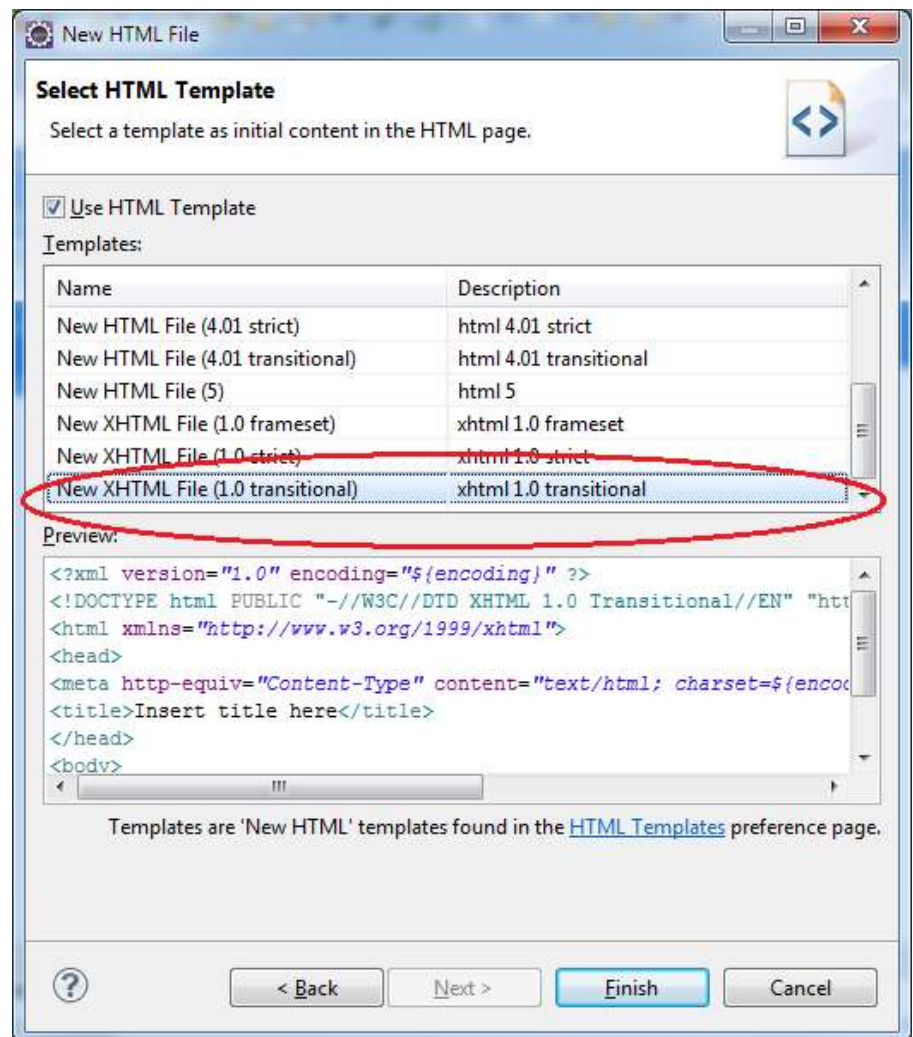
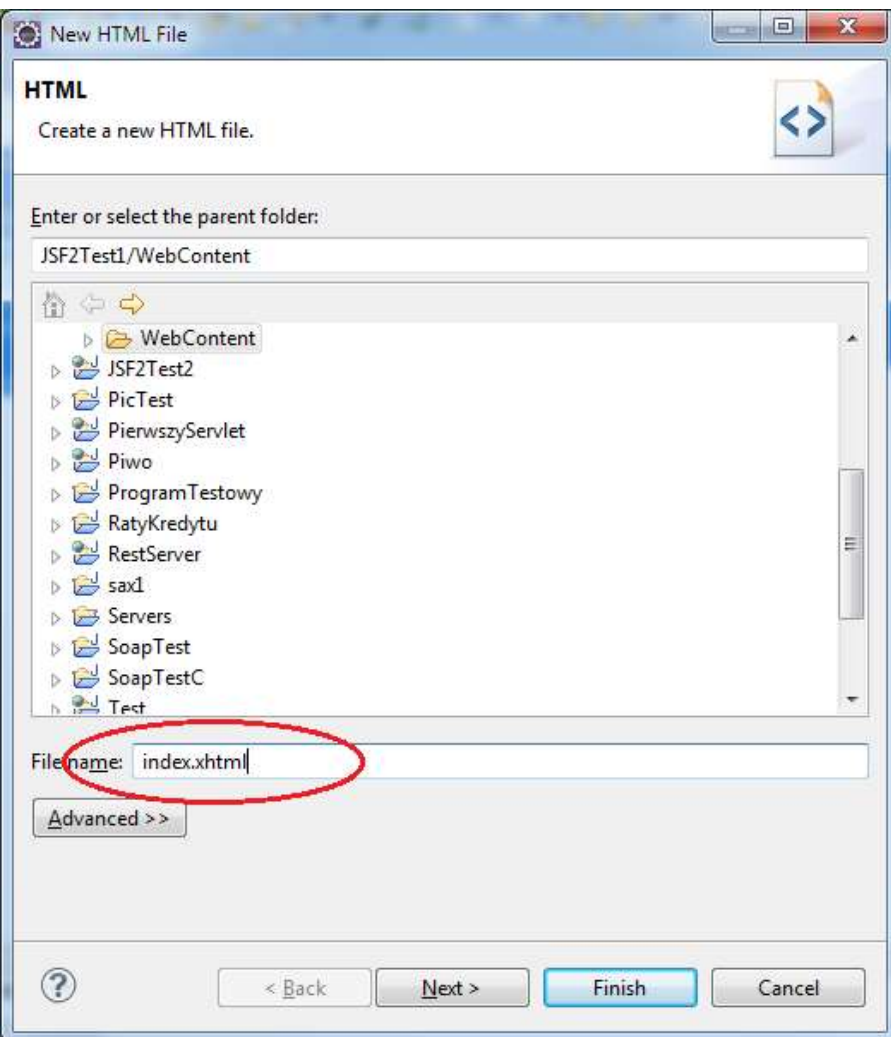
...

```
</faces-config>
```

Większość konfiguracji zamieszczanych
jest w plikach *.java, więc
faces-config.xml może być pusty.



Nowy plik *.xhtml



Struktura pliku index.xhtml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Insert title here</title>
  </h:head>
  <h:body>
    <h1>Sprawdź szansę na wygraną!</h1>
    <h:form >
      Podaj swoje dane:
      PESEL <h:inputText /><br/>
      Imię <h:inputText /><br/>
      Nazwisko <h:inputText /><br/>
      <h:commandButton value="Wyślij" action="#{szansa.wyslij}" />
    </h:form>
  </h:body>
</html>
```

Dodano przestrzeń nazw h

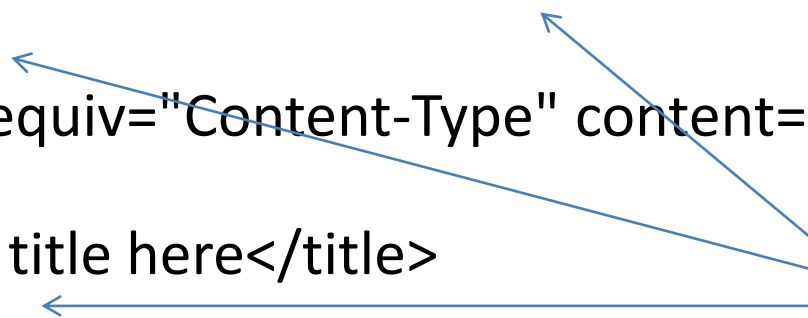
Nie ma konieczności stosować h:body lub h:head. Można używać samych nagłówków body i head. Jednakże przy stosowaniu f:ajax muszą być już tak jak w przykładzie.

Te elementy są ignorowane w tym przykładzie. Powinny one posiadać atrybut value wskazujący na właściwość odpowiedniego beana

Szansa to nazwa klasy (beana). Wyślij to funkcja.

wygrana.xhtml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
    />
    <title>Insert title here</title>
  </h:head>
  <h:body>
    <h1>Wygrałeś!! Gratulacje</h1>
  </h:body>
</html>
```



Tutaj nie ma treści dynamicznych,
ale dobrym zwyczajem jest zawsze
dodawać te elementy JSF 2

przegrana.xhtml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Insert title here</title>
  </h:head>
  <h:body>
    <h:form>
      <h1>Przegrałeś!</h1>
      <p>Spróbuj jeszcze raz <h:link value="kliknij" outcome="index" /> </p>
    </h:form>
  </h:body>
</html>
```

szansa.java

```
import javax.faces.bean.ManagedBean;
```

```
@ManagedBean
```

```
public class Szansa {
```

```
    public String wyslij() {
```

```
        if (Math.random() < 0.2 ) {
```

```
            return "wygrana";
```

```
        } else {
```

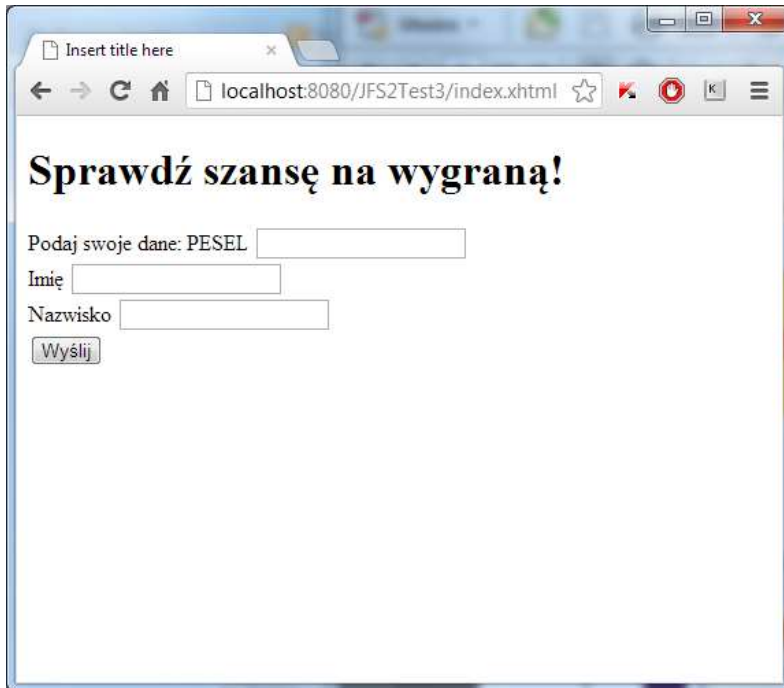
```
            return "przegrana";
```

```
        }
```

```
    }
```

```
}
```

Rezultat



Insert title here

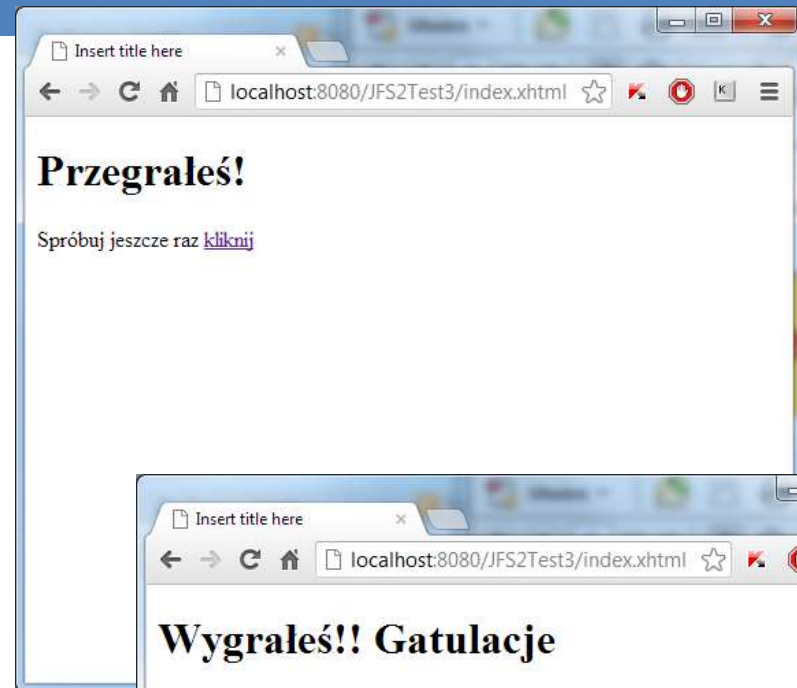
localhost:8080/JFS2Test3/index.xhtml

Sprawdź szansę na wygraną!

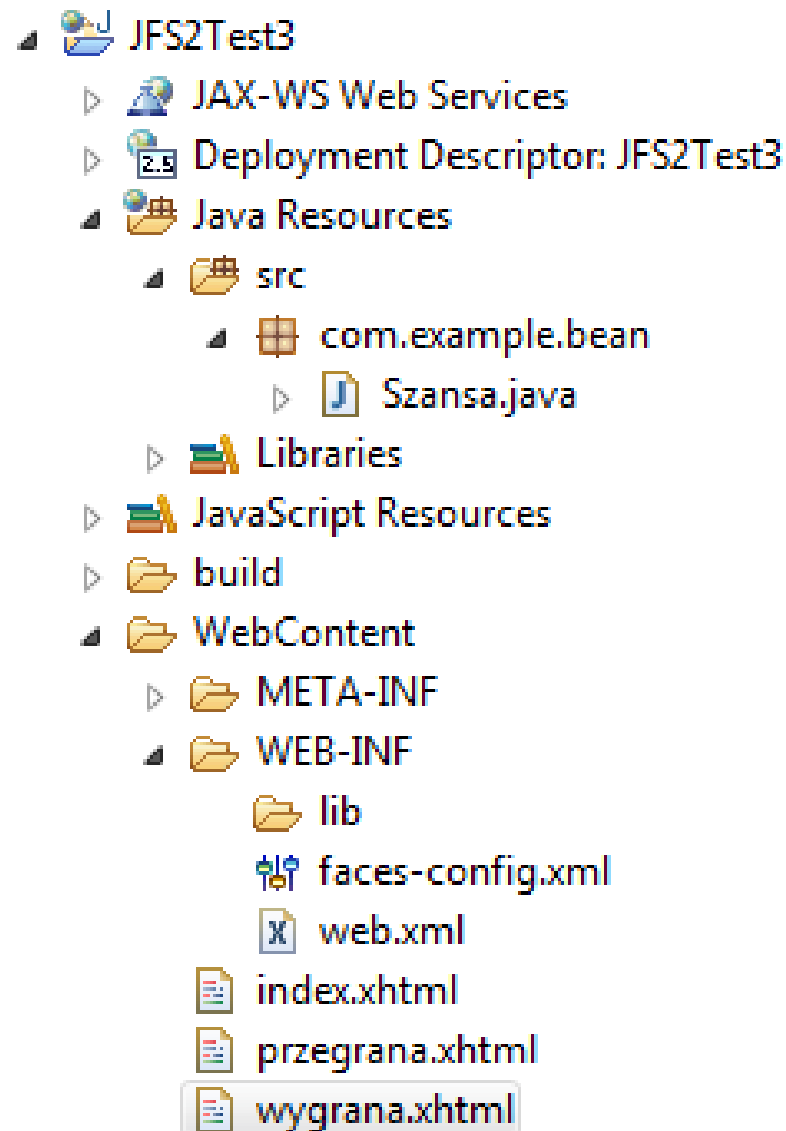
Podaj swoje dane: PESEL

Imię

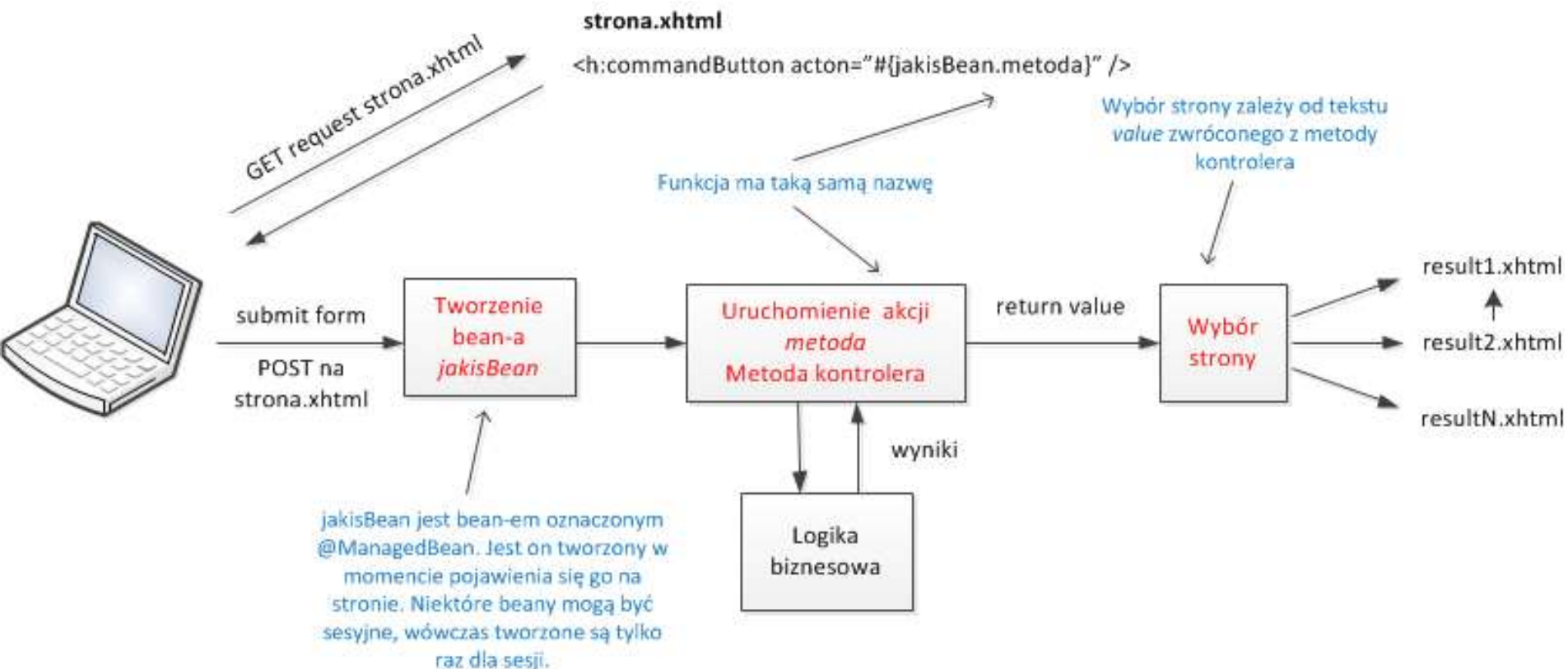
Nazwisko



Rezultat



Przepływ informacji w JSF (wersja bardzo prosta)



@ManagedBean

- Adnotacja @ManagedBeans jest umieszczana przed deklaracją publicznej klasy

@ManagedBean

```
public class SomeName { }
```

- Nazwa klasy (bez nazwy pakietu i z małą pierwszą literą) jest równocześnie nazwą bean-a i pojawia się w wyrażeniu #{someName.cos}
 - Nazwa cos po kropce może oznaczać nazwę funkcji (akcji w przypadku np. h:commandButton) lub właściwości (zmiennej w klasie someName, do której dostęp jest możliwy dzięki funkcją get i set)
- Funkcja akcji kontrolera zwraca String. Jeżeli nie ma ustalonej konfiguracji w pliku faces-config.xml, wówczas tekst ten oznacza nazwy stron xhtml
 - Strony te są w tym samym folderze, gdzie znajdowała się forma

Właściwości

- Wartości wejściowe odpowiadają właściwością bean-a

`<h:inputText value="#{someBean.prop}" />`

- Gdy zawartość formy jest wysyłana (po wykonaniu submit), zawartość z pola tekstowego przekazywana jest jako argument funkcji setProp()
 - Jako pierwsza uruchamiana jest wcześniej walidacja lub konwersja
 - Przy wyświetlaniu formy, wywoływana jest funkcja getProp().
- Domyślnie bean jest o zasięgu request.
 - Bean jest tworzony dwukrotnie – pierwszy raz, gdy formatka jest początkowo wyświetlana i drugi raz, gdy formatka jest submitowana
- Można użyć bezpośrednio wyrażenia `#{someBean.prop}` na stronie html.
 - Wywołana jest zawsze wtedy funkcja getProp()

Przykład – wybor.xhtml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insert title here</title>
</h:head>
<h:body>
<h1>Wybierz język programowania JSF</h1>
<h:form>
    Wpisz nazwę języka programowania JSF:
    <h:inputText value="#{jezykForm.jezyk}" />
    <h:commandButton value="Wybierz" action="#{jezykForm.wybor}" />
</h:form>
</h:body>
</html>
```


Przykład – JezykForm.java

```
import javax.faces.bean.ManagedBean;
```

```
@ManagedBean
```

```
public class JezykForm {
```

```
    private String jezyk;
```

```
    public String getJezyk() {
```

```
        return jezyk;
```

```
    }
```

```
    public void setJezyk(String jezyk) {
```

```
        this.jezyk = jezyk;
```

```
    }
```

LanguageBean.java

```
public String wybor() {  
    if (jezyk==null || jezyk.trim().isEmpty()) {  
        return „pusty-wybor“;  
    } else if (jezyk.equals("Java")) {  
        return „dobry-wybor“;  
    } else {  
        return „zly-wybor“;  
    }  
}  
}
```

pusty-wybor.xhtml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insert title here</title>
</h:head>
<h:body>
    <h1>Nic nie wpisałeś</h1>
    <h:link outcome="wybor" >Spróbuj ponownie</h:link>
</h:body>
</html>
```

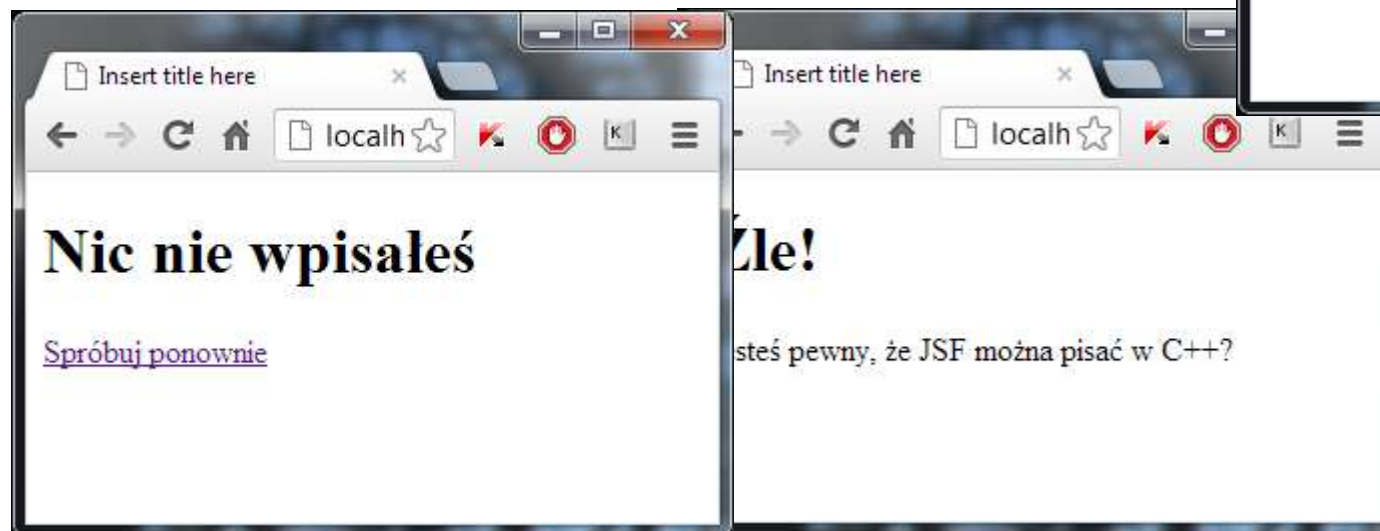
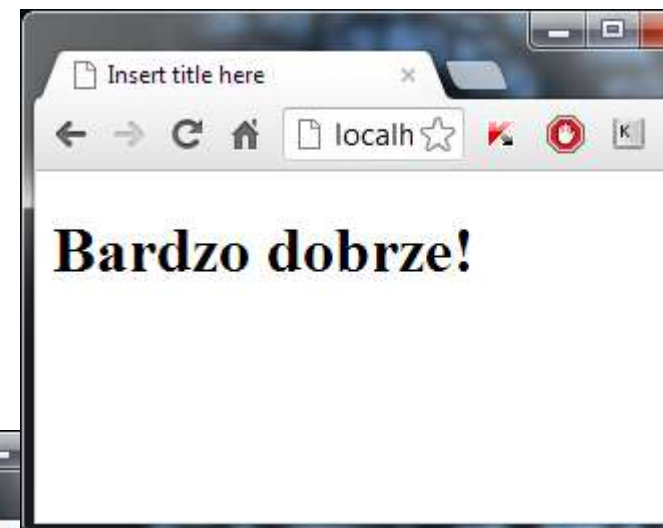
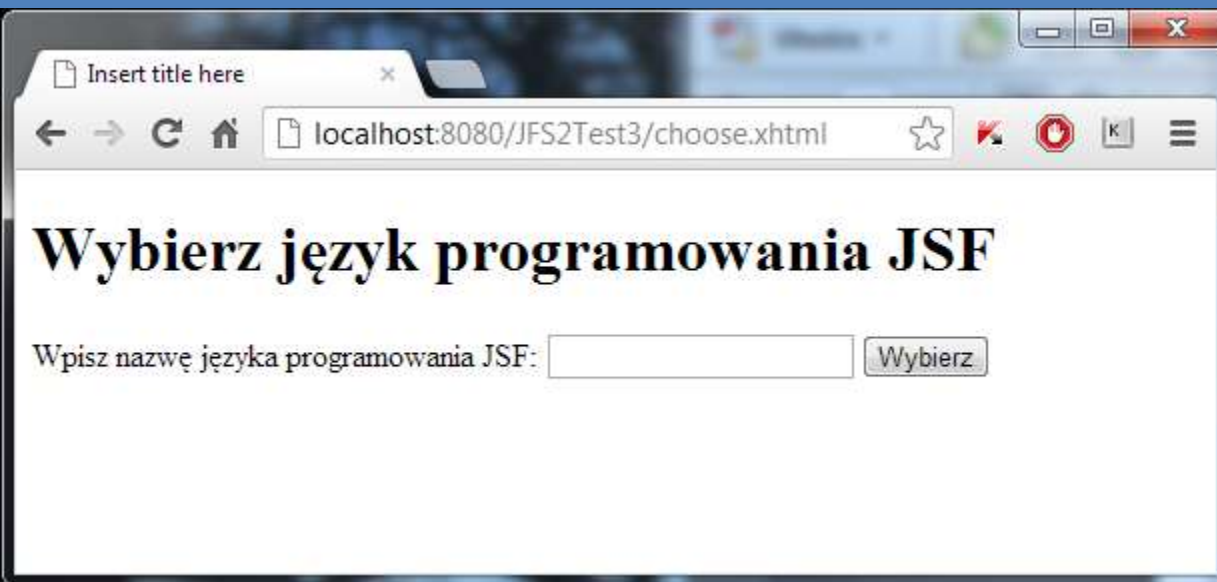
zly-wybor.xhtml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insert title here</title>
</h:head>
<h:body>
  <h1>Źle!</h1>
  <p>Jesteś pewny, że JSF można tworzyć w #{jezykForm.jezyk}?</p>
</h:body>
</html>
```

dobry-wybor.xhtml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Dobry wybór</title>
</h:head>
<h:body>
    <h1>Bardzo dobrze!</h1>
</h:body>
</html>
```

Rezultat



Managed Bean

Managed bean

- Zwykły bean (konstruktor domyślny, gettery i settery, zmienne prywatne) może być Managed beanem
- Składa się z trzech części:
 - Metod dostępu do elementów otrzymywanych z input
 - Akcji kontrolera
 - Miejsca na składowanie rezultatów
- Może służyć do przygotowanie stałych danych wejściowych
 - Dla list rozwijanych (combobox)

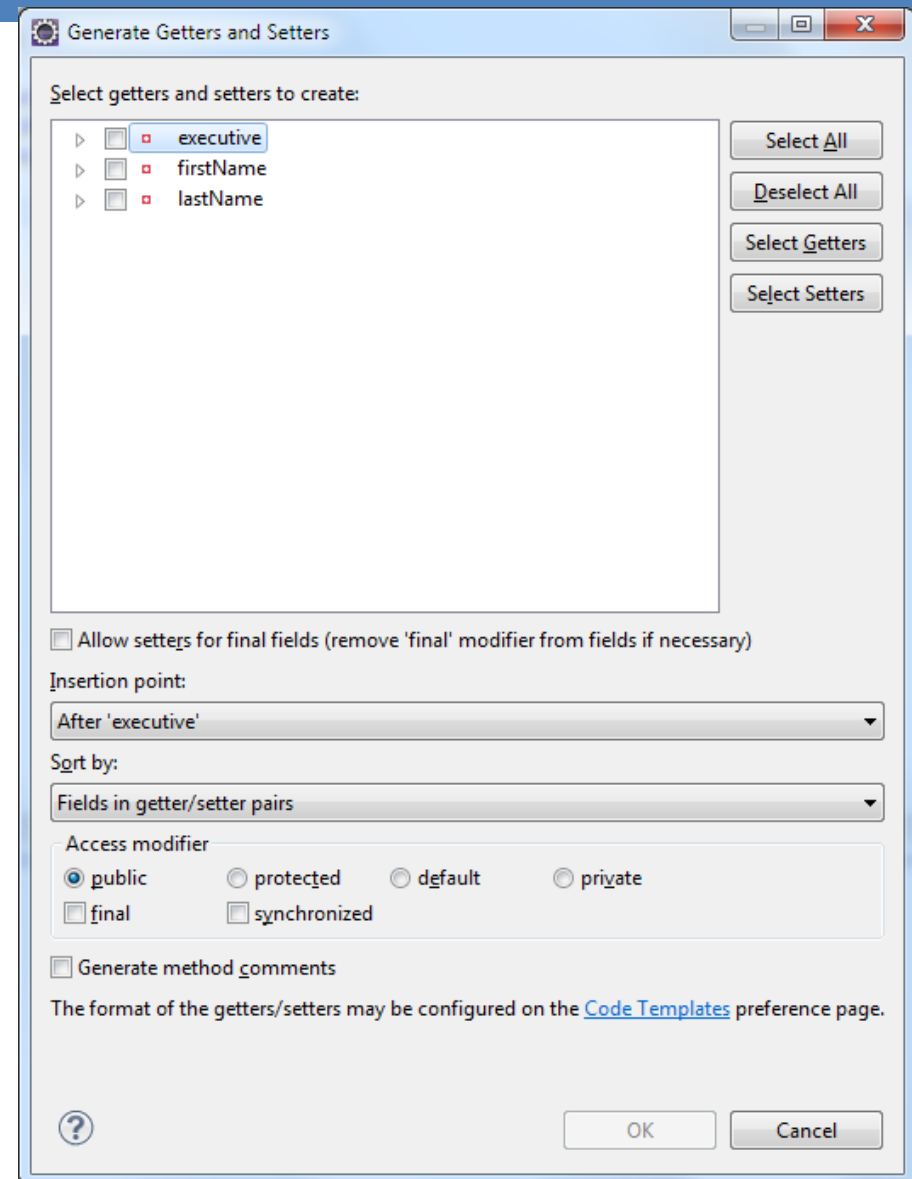
Dostęp do zmiennych

Nazwa metody	Nazwa właściwości	Przykład użycia JSF
getFirstName setFirstName	firstName	<code>{customer.firstName}</code> <code><h:inputText value="{customer.firstName}" /></code>
isExecutive setExecutive (typ boolean)	executive	<code>{customer.executive}</code> <code><h:selectBooleanCheckbox</code> <code>value="{customer.executive}"/></code>
getExecutive setExecutive (typ boolean)	executive	<code>{customer.executive}</code> <code><h:selectBooleanCheckbox</code> <code>value="{customer.executive}"/></code>
getZIP setZIP	ZIP	<code>{address.ZIP}</code> <code><h:inputText value="{address.ZIP}"/></code>

Eclipse – szybkie tworzenie get i set

- W menu kontekstowym wybrać Source/Generate Getters and Setters

```
public class Customer {  
  
    private String firstName;  
    private String lastName;  
    private boolean executive;  
  
}
```



Managed Beans

- JSF zarządza beanami
 - Tworzenie beanów – wymagany jest jedynie bezargumentowy konstruktor
- Kontrolowanie życia beana
 - Zasięg (request, session, application) określa czas życia
- Wywołuje metody setXXX
 - Np. przy `<h:inputText value="#{customer.firstName}" />`, gdy forma jest submitowana, wartość jest przekazywana przez `setFirstName`
- Wywołuje metody getXXX
 - Rezultatem polecenia `#{customer.firstName}` jest wywołanie funkcji `getFirstName`

Problem wydajnościowy funkcji getX

- Problem
 - Metoda get może być wywoływana wielokrotnie
 - Np. za każdym razem, gdy pojawi się `<h:inputText value="#{customer.firstName}" />` lub `#{customer.firstName}` wówczas wywoływana jest metoda `getFirstName`
 - Jeżeli wówczas nawiązywane jest połączenie z bazą danych, wówczas jest to duży problem
- Rozwiązanie
 - Przechowywać wartości w zmiennych, metoda get zwraca aktualną wartość

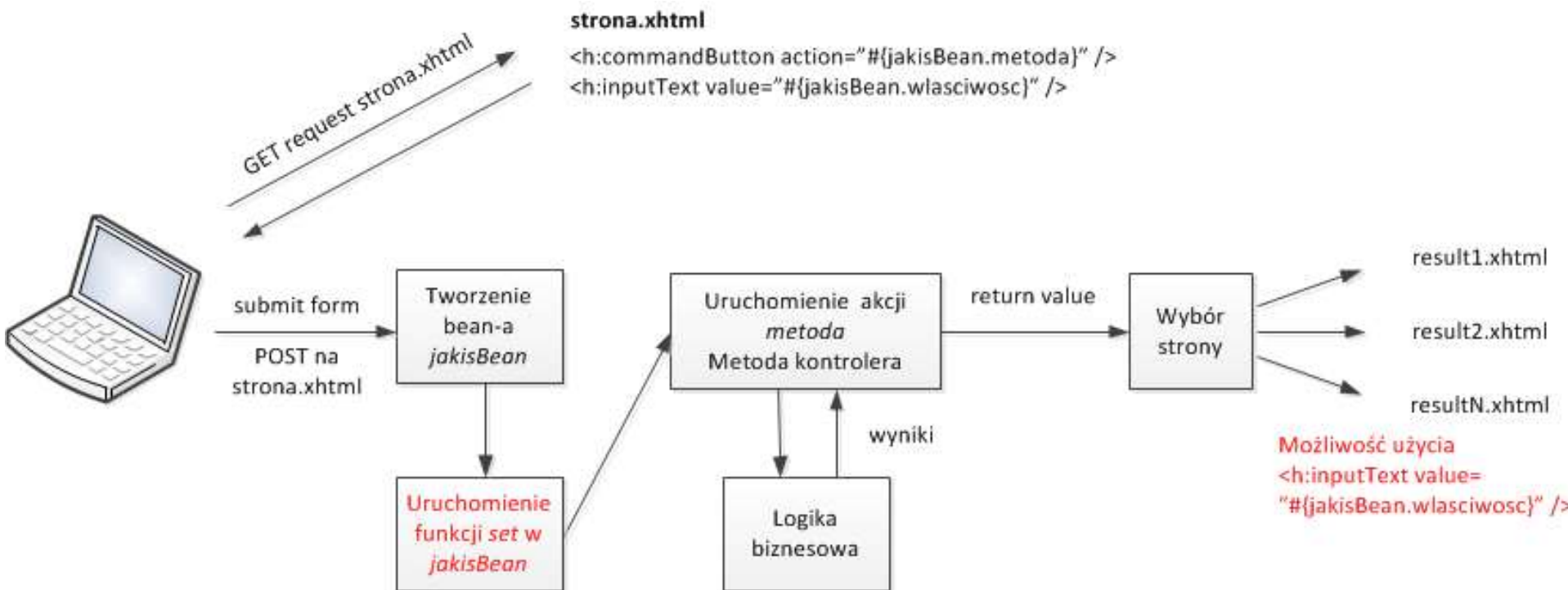
```
private int zmienna = null;
```

```
public int getZmienna() {  
    if (zmienna==null) {  
        zmienna = pobierzZmiennaZBazyDanych();  
        //lub wykonuj skompilowane obliczenia  
    }  
    return zmienna;  
}
```

Trzy części Managed beans

- Właściwości (pary metody getter i setter)
 - Jedna para get i set dla każdego z elementów wejściowych
 - Setter jest automatycznie wywoływany przez JSF gdy forma jest submitowana. Wywołanie to następuje PRZED wykonaniem akcji kontrolera
- Metody kontrolera – akcje
 - Zazwyczaj jest tylko jedna akcja, ale może być kilka, gdy form zawiera wiele przycisków
 - Funkcja akcji (odpowiadająca akcji przywiązanej do przycisku) jest wywoływana automatycznie
- Miejsce dla przechowywania rezultatów
 - Dane nie są wywoływane bezpośrednio przez JSF. Rezultaty tworzone są akcje kontrolera w oparciu o logikę biznesową
 - Dostęp do danych możliwy jest przez funkcję typu getter. Settery nie są potrzebne dla danych tego typu

Rozszerzony przepływ informacji (wciąż uproszczony)



Przykładowa aplikacja

- Założenia
 - Wpisywany jest numer klienta i hasło
 - Otrzymywane informacje:
 - Strona prezentująca imię, nazwisko i stan konta (dwie wersje w zależności od stanu konta - ujemne i dodatnie)
 - Strona błędu, w przypadku niepoprawnych danych
- Założenia Managed bean
 - Właściwości odpowiadające danym wejściowym
 - idKlienta, hasło
 - Metody akcji
 - Kojarzy idKlienta z Klientem i zachowuje Klienta jako zmienną
 - Miejsce na rezultat
 - Początkowa zmienna przechowująca dane Klienta jest pusta. Dostęp jedynie przez getter

bank-login.xhtml – dane wejściowe

```
<fieldset>
<legend> Zaloguj się</legend>
<h:form>
```

Id klienta:

```
<h:inputText value="#{bankingBean.idKlienta}"/>
<br/>
```

Hasło:

```
<h:inputSecret value="#{bankingBean.haslo}"/> <br/>
<h:commandButton value="Pokaż saldo"
action="#{bankingBean.pokazSaldo}"/>
```

```
</h:form>
</fieldset>
```

Ta wartość odgrywa podwójną rolę. Gdy strona jest pierwszy raz wyświetlana, wywoływana jest funkcja `getIdKlienta`. Jeżeli wartość nie jest pustą, wyświetlana w tym miejscu.
W przypadku submit, wartość przekazywana jest metodą `setIdKlienta` do beana.



BakingBean.java – właściwości dla danych wejściowych

```
private String idKlienta, haslo;

public String getIdKlienta() {
    return idKlienta;
}

public void setIdKlienta(String idKlienta) {
    this.idKlienta = idKlienta;
    if (idKlienta.isEmpty()) {
        this.idKlienta = "brak danych";
    }
}

public String getHaslo() {
    return haslo;
}

public void setHaslo(String haslo) {
    this.haslo = haslo;
}
```

BakingBean.java – metody akcji

```
private static KlientLookupService klientService = new KlientSimpleMap();
```

```
public String pokazSaldo() {  
    if (!haslo.equals("test")) {  
        return "bank-zle-haslo";  
    }  
    klient = klientService.findKlient( idKlienta );  
    if (klient==null) {  
        return "bank-nieznany-klient";  
    } else if (klient.getSaldo() < 0 ) {  
        return "bank-ujemne-saldo";  
    } else {  
        return "bank-normalne-saldo";  
    }  
}
```

Te zmienne są
wypełnione przed
wywołaniem tej
funkcji.

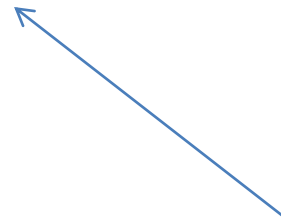
BankingBean.java – dane rezultatów

```
private Klient klient;
```



Wypełniana jest przez kontroler

```
public Klient getKlient() {  
    return klient;  
}
```



Funkcja potrzebna, aby
dostać się do składowych
klasy Klient, np. tak:
#{bankingBean.klient.imie}

Logika Biznesowa

```
public interface KlientLookupService {  
  
    public Klient findKlient(String id);  
  
}
```

Logika biznesowa - implementacja

```
public class KlientSimpleMap implements KlientLookupService {
```

```
    private Map<String, Klient> klienci;
```

```
    public KlientSimpleMap() {
```

```
        klienci = new HashMap<String, Klient>();
```

```
        Klient k1 = new Klient("Alicja", "Kowalska", "id01", -3453.44 );
```

```
        Klient k2 = new Klient("Anna", "Nowak", "id02", 1234.56 );
```

```
        Klient k3 = new Klient("Jerzy", "Kamiński", "id03", 987654.32 );
```

```
        klienci.put(k1.getId(), k1);
```

```
        klienci.put(k2.getId(), k2);
```

```
        klienci.put(k3.getId(), k3);
```

```
    }
```

```
    public Klient findKlient(String id) {
```

```
        if (id != null) {
```

```
            return klienci.get(id);
```

```
        } else {
```

```
            return null;
```

```
        }
```

```
    }
```

```
}
```

bank-normalne-saldo.xhtml

```
<ul>  
  <li>Imie: #{bankingBean.klient.imie}</li>  
  <li>Nazwisko: #{bankingBean.klient.nazwisko}</li>  
  <li>Id: #{bankingBean.klient.id}</li>  
  <li>Saldo: #{bankingBean.klient.saldo}</li>  
</ul>
```

bank-nieznany-klient.xhtml

<p>Nieznany klient #{bankingBean.idKlienta}</p>

<p>Spróbuj ponownie</p>

Kontrolki

- Pole tekstowe
 - `<h:inputText value="#{bean.wartosc}" />`
 - Pole domyślnie jest wypełnione zawartością właściwości pobranej funkcją `get`.
- Checkbox
 - `<h:selectBooleanCheckbox value="#{bean.wartosc}" />`
 - Gdy funkcja `get` właściwości zwraca `true`, wówczas checkbox jest zaznaczony
- Combobox, lista rozwijana
 - `<h:selectOneMenu value="#{bean.wartosc}" >`
 - Gdy wartość zwrócona przez funkcję `get` odpowiada wartości z listy, jest ona zaznaczona

Lista rozwijana

- Przykład

```
<h:selectOneMenu value="#{jakisBean.wartosc}">  
    <f:selectItems value="#{jakisBean.mozliwosci}"/>  
</h:selectOneMenu>
```

- Gdy zawartość strony jest generowana, wówczas wywołana jest funkcja `jakisBean.getMozliwosci`. Zwraca ona zawartość rozwijanej listy.
- Następnie wywołana jest funkcja `getWartosc`. Jeżeli jej wartość znajduje się na liście, jest ona zaznaczona.
- Zmienna `mozliwosci` w bean powinna być typu `List<SelectItem>` lub `SelectItem[]`.

Przykład – strona z h:selectOneMenu

```
<!DOCTYPE ...>  
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:f="http://java.sun.com/jsf/core"  
      xmlns:h="http://java.sun.com/jsf/html">  
<h:head>  
...  
</h:head>
```



Przestrzeń nazw dla <f:selectItems

Przykład – strona z h:selectOneMenu

```
<h:form>
    Wybierz język programowania:
    <h:selectOneMenu value="#{wyborJęzyka.język}">
        <f:selectItems value="#{wyborJęzyka.listaJęzykow}"/>
    </h:selectOneMenu>
    <h:commandButton value="Wybierz"          action="#{wyborJęzyka.wybierz}"
/>
</h:form>
```

Przykład – strona z h:selectOneMenu

@ManagedBean

```
public class WyborJęzyka {
```

```
    private List<SelectItem> lista = null;
```

```
    public List<SelectItem> getListaJęzykow() {
```

```
        if (lista == null) {
```

```
            lista = new ArrayList<SelectItem>();
```

```
            lista.add(new SelectItem("C++", "Język C++"));
```

```
            lista.add(new SelectItem("Java", "Język Java"));
```

```
            lista.add(new SelectItem("C#", "Język C#"));
```

```
        }
```

```
        return lista;
```

```
    }
```

Przykład strona z h:selectOneMenu

