Podstawy technologii JavaServer Faces

wg

https://docs.oracle.com/javaee/7/JEETT.pdf

rozdział 8

Wykład 3

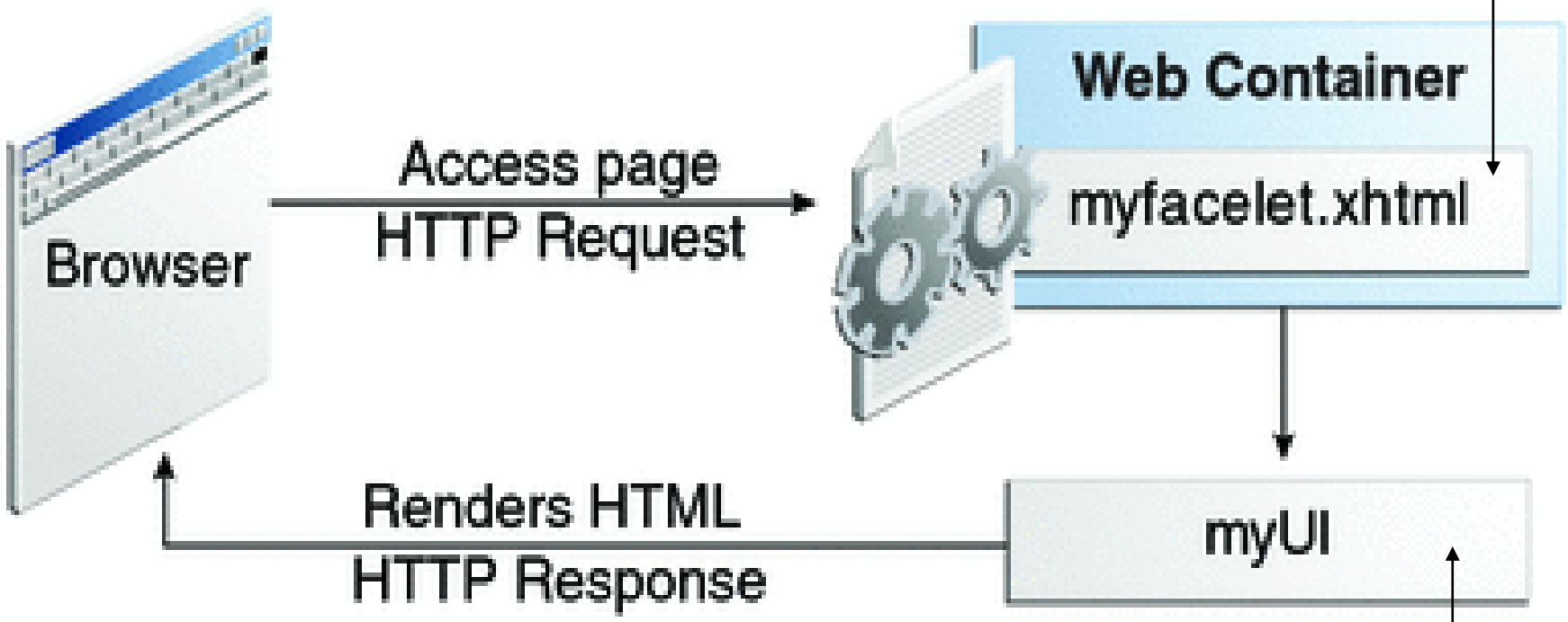
Technologie internetowe

Z czego składa się technologia JavaServer Faces ? (wykład 2, str.2-3, 7)

- interfejsu programowania aplikacji internetowej reprezentujący komponenty i zarządzanie ich stanem; obsługi zdarzeń, walidacji po stronie serwera, konwersji danych; nawigacji stron; wspieranie internacjonalizacji i dostępności; zapewnienia rozszerzalność wszystkim tym elementom
- biblioteki znaczników do dodawania komponentów do stron internetowych i obiektów po stronie serwera

Interakcja żądanie - odpowiedź między warstwą klienta i warstwą internetową w typowej aplikacji JavaServer Faces (wykład 2, str.5-8)

znaczniki komponentów JavaServer Faces, odniesienia do słuchaczy zdarzeń, walidatorów oraz konwerterów, komponentów JavaBeans, pozyskujące i przetwarzające dane specyficznie dla komponentów



Komponenty JavaServer Faces używane w widokach (UI)

Przebieg fazy: żądanie - odpowiedź

Strona JSF - myfacelet.xhtml, **jest zbudowana ze znaczników komponentów JavaServer Faces**.

Rola znaczników:

- łączą widok z **komponentami widoku** (reprezentowanymi przez myUI na rysunku), które są po stronie serwera, w warstwie internetowej
- łączą ze **słuchaczami zdarzeń, walidatorami oraz konwerterami**, które są reprezentowane przez komponenty
- łączą z **komponentami JavaBeans**, pozyskującymi i przetwarzającymi dane specyficznie dla komponentów

Skutkiem żądania z warstwy klienta (**Request**), widok jest renderowany jako odpowiedź (**Response**).
Renderowanie jest procesem, w którym na podstawie zawartości strony kontener internetowy tworzy strony typu HTML lub XHTML, które mogą być odczytywane przez warstwę klienta zawierającą przeglądarkę.

Technologia JavaServer Faces wspiera budowę aplikacji wielowarstwowej

- **Separacja zachowania i prezentacji** w aplikacji internetowej dzięki odwzorowaniu żądania HTTP na specyficzną dla komponentu obsługę zdarzeń oraz zarządzanie komponentami jako obiektami o określonym czasie życia (stateful) po stronie serwera
- **Separacja logiki biznesowej od prezentacji** pozwala programistom stron internetowych posługiwać się jedynie językiem znaczników bez konieczności używania języka proceduralnego Java
- **Możliwość zastosowania różnych implementacji języków skryptowych** dzięki używaniu API technologii JSF bezpośrednio przez API Java Servlet.

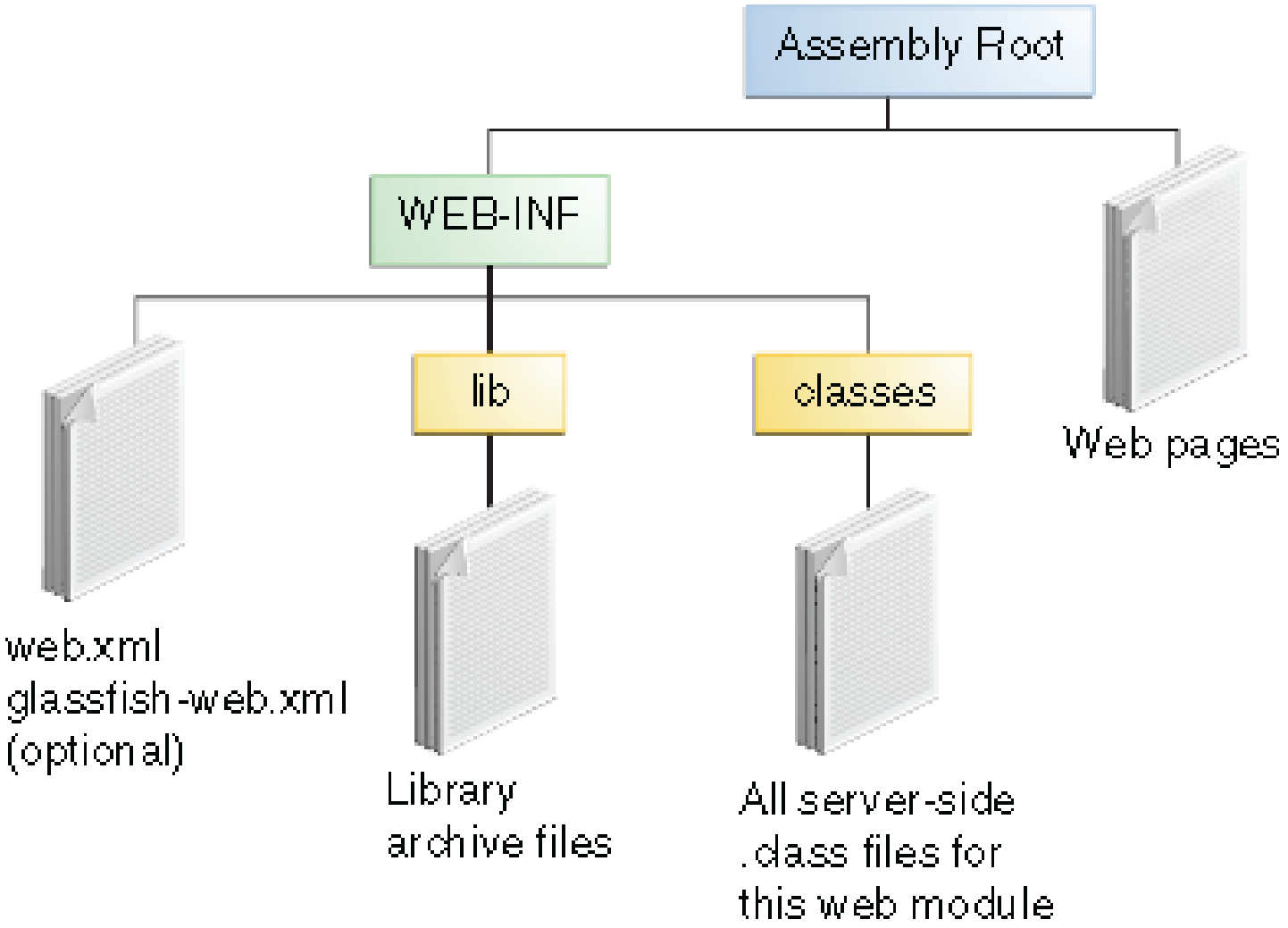
Zadania wykonywane podczas tworzenia aplikacji internetowej

- Utworzenie strony internetowej typu JSF
- Wstawienie komponentów do strony internetowej za pomocą wstawienia ich znaczników
- Powiązanie komponentu na stronie internetowej do danych po stronie serwera
- Powiązanie zdarzeń generowanych przez komponenty do kodu aplikacji po stronie serwera
- Zachowanie i odtwarzanie stanu aplikacji podczas cyklu życia żądania wysłanego do serwera
- Ponowne wykorzystanie komponentów i rozszerzanie ich własności

Co zawiera aplikacja typu JavaServer Faces (wykład1, str.27-28)

- **Zbiór stron internetowych** zawierających znaczniki komponentów
- **Zbiór znaczników umożliwiających umieszczenie komponentów na stronie internetowej**
- **Zbiór obiektów typu Managed Bean,** które są obiektami zarządzanymi przez kontener internetowy, z minimalnymi wymaganiami. Wspierają one injekcję zasobów oraz akcje występujące w cyklu życia żądanie-odpowiedź
- **Deskryptor wdrożenia web.xml**
- **Opcjonalnie, pliki konfiguracji zasobów aplikacji np. faces-config.xml:** reguły nawigacji stron internetowych, konfiguracja ziaren oraz niestandardowych obiektów i komponentów,
- **Opcjonalnie, zbiór obiektów niestandardowych,** które zawierają komponenty niestandardowe, walidatory, konwertery lub słuchacze zdarzeń, tworzone przez programistów
- **Opcjonalnie, zbiór znaczników niestandardowych** reprezentujących obiekty niestandardowe na stronie

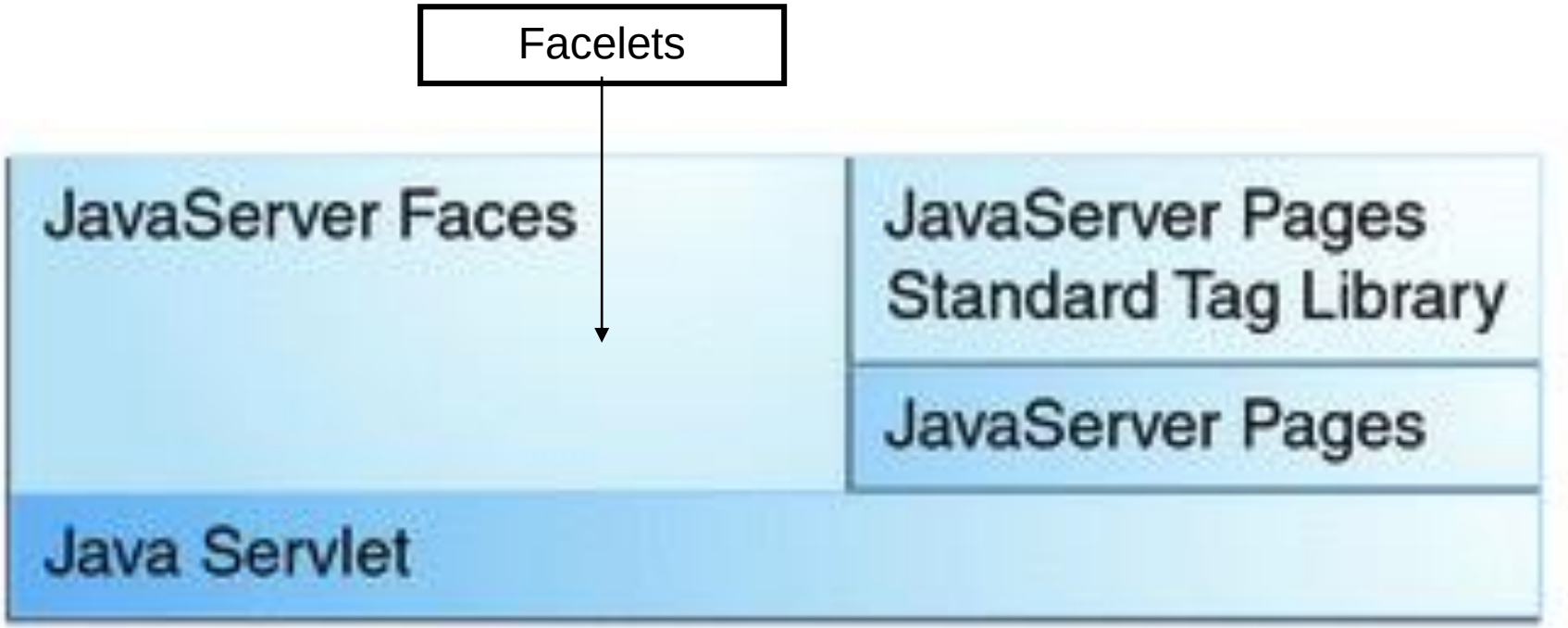
Struktura modułu internetowego (wykład 1, str.27-28)



Język znaczników Facelets

- Użycie XHTML do tworzenia strony internetowej
- Korzystanie z biblioteki znaczników Facelets, JavaServer Faces i JSTL
- Korzystanie z języka Expression Language (EL)
- **Elementy wspierające budowę dużej aplikacji:**
  - Stosowanie szablonów komponentów i stron umożliwia **wielokrotne używanie kodu**
  - Funkcjonalne **rozszerzanie właściwości komponentów** i obiektów po stronie serwera wspierane stosowaniem **adnotacji**
  - **Bogata architektura** do zarządzania komponentami, przetwarzania danych komponentów, walidacji danych użytkownika aplikacji oraz obsługi zdarzeń
  - **Krótki czas kompilacji**
  - Walidacja wyrażeń języka EL podczas kompilacji
  - **Wysoka wydajność renderowania** stron aplikacji

Warstwowa struktura API



Opis znaczników obsługiwanych przez Facelets

- Znaczniki do tworzenia struktury strony (komponenty UI strony)
- Znaczniki do tworzenia szablonów strony

Biblioteki znaczników obsługiwanych przez Facelets

Biblioteki znaczników	URI	Prefiks	Przykład	Zawartość
JavaServer Faces Facelets Tag Library	http://xmlns.jcp.org/jsf/facelets	ui:	ui:component ui:insert	Znaczniki szablonów
JavaServer Faces HTML Tag Library	http://xmlns.jcp.org/jsf/html	h:	h:head h:body h:outputText h:inputText	Znaczniki komponentów JavaServer Faces dla wszystkich obiektów komponentów UI
JavaServer Faces Core Tag Library	http://xmlns.jcp.org/jsf/core	f:	f:actionListener f:attribute	Znaczniki niestandardowych akcji JavaServer Faces, które są niezależne od narzędzia renderowania
JSTL Core Tag Library	http://xmlns.jcp.org/jsp/jstl/core	c:	c:forEach c:catch	JSTL 1.2 Core Tags
JSTL Functions Tag Library	http://xmlns.jcp.org/jsp/jstl/functions	fn:	fn:toUpperCase fn:toLowerCase	JSTL 1.2 Functions Tags

Biblioteki znaczników obsługiwanych przez Facelets (cd)

Biblioteki znaczników	URI	Prefiks	Przykład	Zawartość
Pass-through Elements Tag Library	**http://xmlns.jcp.org/jsf**	**jsf:**	jsf:id	Znacznik wspierający odwzorowanie do znaczników HTML5
Pass-through Elements Tag Library	**http://xmlns.jcp.org/jsf/passthrough**	**p:**	p:type	Znacznik wspierający odwzorowanie do znaczników HTML5
Composite Component Tag Library	**http://xmlns.jcp.org/jsf/composite**	**cc:**	cc:interface	Znaczniki wspierające komponenty kompozytowe

Opis znaczników obsługiwanych przez Facelets (tutorial EE 7)

rozdziały 10 -11

Przegląd znaczników JSF (UI) - composite

Znacznik	Funkcja znaczników szablonu
composite:interface	Definicja jednego komponentu jako połączenie cech wielu komponentów
composite:implementation	Definiuje implementację kompozytowego komponentu. W przypadku definicji **composite:interface** implementacja musi być zgodna z tą definicją
composite:attribute	Deklaracja atrybutu instancji komponentu, do którego ten znacznik jest przypisany
composite:insertChildren	Dowolny komponent lub tekst szablonu ze znacznikiem kompozytowym w używanej stronie jest powtarzany w punkcie umieszczenia tego znacznika w ramach znacznika **composite:implementation**

Przegląd znaczników JSF (UI) - composite (cd)

composite:valueHolder	Deklaracja znacznika wewnątrz znacznika **composite:interface**. Definicja implementacji **ValueHolder** właściwego dla obiektów używanych na stronie
composite:editableValueHolder	Deklaracja znacznika wewnątrz znacznika **composite:interface**. Definicja implementacji **EditableValueHolder** właściwego dla obiektów używanych na stronie
composite:actionSource	Deklaracja znacznika wewnątrz znacznika **composite:interface**. Definicja implementacji **actionSource** właściwego dla obiektów używanych na stronie

Przegląd znaczników JSF (UI) - h

Znacznik	Funkcja	Renderowany	Widok
h:body		body	Definicja zawartości strony
h:button	Przesyła dane z formularza do aplikacji	Element HTML5 <button>	Przycisk
h:commandLink	Łączy z inną stroną lub z innym miejscem danej strony	Element HTML <a >	Hyperlink
h:dataTable	Reprezentuje widok zbioru złożonych danych	Element HTML <table>	Tablica modyfikowana dynamicznie
h:form	Reprezentuje formularz danych wejściowych, które mogą być przesłane razem do aplikacji	Element HTML <form>	Brak widoku
h:column	Reprezentuje kolumnę danych w komponencie	Kolumna w tabeli HTML	Kolumna w tabeli

Przegląd znaczników JSF (UI) – h (cd)

Znacznik	Funkcja	Renderowany jako	Widok
h:commandButton	Przesyła dane z formularza do aplikacji	Element HTML <input type=*type*> gdzie type może być: "submit", "reset", "image"	Przycisk
h:graphicImage	Wyświetla obraz	Element HTML <img>	Obraz
h:inputHidden	Pozwala autorowi strony na ukrycie elementu strony	Element HTML <input type="hidden">	Brak widoku
h:inputText	Wprowadzanie danych	Element HTML <input type="text">	Pole tekstowe
h:inputTextarea	Pozwala wprowadzać łańcuch wielowierszowy	Element HTML <textarea>	Pole wielowierszowe

Przegląd znaczników JSF (UI) – h (cd)

Znacznik	Funkcja	Renderowany jako	Widok
h:inputSecret	Pozwala użytkownikowi wprowadzić łańcuch bez pokazania jego zawartości	Element HTML <input type="password">	Pole tekstowe zawierające znaki maskujące aktualną zawartość wprowadzonego łańcucha
h:message	Wyświetla komunikat	Znacznik HTML jeśli zastosowano styl	Łańcuch tekstu
h:messages	Wyświetla komunikaty	Zbiór znaczników HTML , jeśli zastosowano styl	Łańcuch tekstu
h:outputLink **h:link**	Łączy z inna stroną lub położeniem na stronie bez generowania zdarzenia	Element HTML <a>	Hyperlink

Przegląd znaczników JSF (UI) – h (cd)

Znacznik	Funkcja	Renderowany jako	Widok
h:outputFormat	Wyświetla komunikat	Zwykły tekst	Zwykły tekst
h:outputLabel	Wyświetla zagnieżdżony komponent jako etykieta podanego pola wejściowego	Element HTML <label>	Zwykły tekst
h:outputText	Wyświetla linię tekstu	Zwykły tekst	Zwykły tekst
h:panelGrid	Wyświetla tabelę	Elementy HTML: <table> z <tr> i <td>	Tabela
h:panelGroup	Grupuje komponenty za pomocą jednego nadrzędnego komponentu	Element HTML lub	Wiersz tabeli
h:selectBooleanCheckbox	Pozwala zmienić wartość za pomocą wyboru typu Boolean	Element. HTML <input type="checkbox">	Przycisk wyboru

Przegląd znaczników JSF (UI) – h (cd)

Znacznik	Funkcja	Renderowany	Widok
h:selectManyCheckbox	Wyświetla zbiór przycisków wielokrotnego wyboru	Zbiór elementów HTML <input> typu przycisk wyboru	Zbiór przycisków wyboru
h:selectManyListbox	Możliwy wyboru wielu pozycji wyświetlanej listy	Element HTML <select>	lista
h:selectManyMenu	Wyświetla listę menu, gdzie można wybrać wiele pozycji	Element HTML <select>	Przewijany widok typu ComboBox
h:selectOneListbox	Wyświetla listę, gdzie można dokonać wyboru jednej pozycji listy	Element HTML <select>	lista
h:selectOneMenu	Wyświetla listę menu, gdzie można jedną pozycję	Element HTML <select>	Przewijany widok typu ComboBox
h:selectOneRadio	Wyświetla zbiór przycisków, gdzie można dokonać wyboru jednego przycisku	Element HTML <input type="radio">	Zbiór przycisków typu „radio”

1-y sposób - przegląd znaczników JSF odwzorowanych do znaczników HTML5 (2014) wynikających z zastosowanych typów atrybutów

http://www.w3schools.com/html/html5_intro.asp

Table 8–4 How Facelets Renders HTML5 Elements

HTML5 Element Name	Identifying Attribute	Facelets Tag
a	jsf:action	h:commandLink
a	jsf:actionListener	h:commandLink
a	jsf:value	h:outputLink
a	jsf:outcome	h:link
body		h:body
button		h:commandButton
button	jsf:outcome	h:button
form		h:form
head		h:head
img		h:graphicImage

1-y sposób - przegląd znaczników JSF odwzorowanych do znaczników HTML5 wynikających z zastosowanych typów atrybutów (2)

input	type="button"	h:commandButton
input	type="checkbox"	h:selectBooleanCheckbox
input	type="color"	h:inputText
input	type="date"	h:inputText
input	type="datetime"	h:inputText
input	type="datetime-local"	h:inputText
input	type="email"	h:inputText
input	type="month"	h:inputText
input	type="number"	h:inputText
input	type="range"	h:inputText
input	type="search"	h:inputText
input	type="time"	h:inputText

1-y sposób - przegląd znaczników JSF odwzorowanych do znaczników HTML5 wynikających z zastosowanych typów atrybutów (3)

input	type="url"	h:inputText
input	type="week"	h:inputText
input	type="file"	h:inputFile
input	type="hidden"	h:inputHidden
input	type="password"	h:inputSecret
input	type="reset"	h:commandButton
input	type="submit"	h:commandButton
input	type="*"	h:inputText
label		h:outputLabel
link		h:outputStylesheet
script		h:outputScript
select	multiple="*"	h:selectManyListbox
select		h:selectOneListbox
textarea		h:inputTextArea

2-i sposób - przykład atrybutów bezpośrednio odwzorowanych do znaczników HTML5 (3)

Przed renderowaniem:

```
<html ... xmlns:p="http://xmlns.jcp.org/jsf/passthrough"
...
<h:form prependId="false">
<h:inputText id="nights" p:type="number" value="#{bean.nights}"
    p:min="1" p:max="30" p:required="required"
    p:title="Enter a number between 1 and 30 inclusive.„
>
```

Po renderowaniu:

```
<input id="nights" type="number" value="1" min="1" max="30"
     required="required"
      title="Enter a number between 1 and 30 inclusive."
>
```

Najczęściej występujące atrybuty w znacznikach komponentów

Atrybut	Opis
binding	Identyfikuje właściwość obiektu (atrybut i jego metody typu set i get lub same metody) i binduje do niej komponent
id	Identyfikuje komponent
immediate	Jeśli ma wartość true, zdarzenia, walidacje i konwersje związane z komponetem powinny się odbywać, kiedy rozpoczyna się obsługa fazy żądania
rendered	Specyfikuje warunek renderowania komponentu. Jeśli warunek nie jest spełniony, komponent nie jest renderowany.
style	Specyfikuje Kaskadowy arkusz stylu (CSS) znacznika np. **style="border:solid 1px"**
styleClass	Specyfikuje nazwę klasę CSS, która zawiera definicję stylów np. klasa **.center_content** z pliku **cssLayout.css**
value	Specyfikuje wartość komponentu w postaci wyrażenia

Atrybut id

Używany w przypadku powiązania z innym komponentem lub klasą po stronie sewera. W przypadku braku deklaracji takiego atrybutu, implementacja JSF generuje automatycznie atrybut id.

Atrybut immediate

Komponenty wejściowe i komponenty generujące polecenia (implementujące interfejs **javax.faces.component.ActionSource** np. przyciski, hyperlinki) przy wartości atrybutu **true** mogą obsługiwać zdarzenia, walidację i konwersję, na początku cyklu życia JSF. Należy ustawić we wszystkich komponentach ten atrybut na **true**, jeśli są powiązane logicznie podczas obsługi tych zdarzeń.

Atrybut rendered

Wartość tego atrybutu decyduje o możliwości umieszczenia widoku danego komponentu na stronie zwracanej do przeglądarki.

Atrybuty style i styleClass

Umożliwiają specyfikację kaskadowego arkusza stylu http://www.w3.org/Style/CSS/

Atrybuty value i binding

Atrybuty te wiążą komponent z danymi obiektowymi (komponentu typu **ManagedBean**)

(1) Dodawanie wybranych znaczników html i body (przykłady: 1 (wykład1), 2 (wykład 2))

Definicje strony xhtml w technologii JSF:

```
<!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://xmlns.jcp.org/jsf/html">
   <h:head> Add a title
   </h:head>
   <h:body> Add Content
   </h:body>
</html>
```

Strona xhtml po renderowaniu

```
<!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">
   <head>
      <title>Add a title</title>
   </head>
     <body> Add Content
     </body>
  </html>
```

(2) Dodawanie wybranych znaczników do strony form

Dodawanie komponentu Form (wykład 2, przykład 2)

<h:form>

inne znaczniki reprezentujące elementy formularza

</h:form>

Zastosowanie komponentów typu text

- Etykiety (labels)
- Pola tekstowe
- Obszary tekstowe
- Pole tekstowe hasła maskujące wprowadzane litery

(3) Dodawanie wybranych znaczników (wykład 2, przykład 2)

Znaczniki pól wejściowych tekstowych

h:inputHidden
h:inputSecret
h:inputText
h:inputTextarea

Wybrane atrybuty znaczników typu input	**Opis**
id	Unikalny identyfikator komponentu w zasięgu komponentu - właściciela
value	Wartość przechowująca wprowadzone dane
converter	Określa typ konwertera wartości wprowadzonej do komponentu lub wyprowadzania
converterMessage	Definicja wiadomości określającej błędy konwersji
dir	Specyfikuje kierunek wyświetlania tekstu LTR (od do prawej) i RTL (od prawej do lewej)
label	Specyfikacja nazwy komponentu używana w wiadomościach o błędach

(3cd) Dodawanie wybranych znaczników do strony

Atrybuty znaczników typu input cd.	**Opis**
lang	Specyfikuje kod związany z podanym językiem np. pl
requiredMessage	Definicja wiadomości określającej brak danej
value	wartość wprowadzona do komponentu
required	Wartość **true** oznacza konieczność wprowadzenia danej do pola komponentu (value)
validator	Identyfikuje metodę ziarna typu Managed Bean do obsługi walidacji danego komponentu (inny sposób: zagnieżdżenie znacznika **f:validator**)
validatorMessage	Definicja wiadomości określającej brak poprawnego wyniku walidacji
valueChangeListener	Identyfikuje metodę, która obsługuje zdarzenie wprowadzenia danej do komponentu

Przykład 3 - Dodawanie atrybutu validator do znacznika input (rozbudowa przykładu 2 z wykładu 2)

```
<h:inputText
    id="name"
    title="Imie: "
    value="#{personalia.name}"
    required="true"
    requiredMessage="Blad: Podaj imie."
    maxlength="10"
    validator="#{personalia.validateName}"
    />
```

Metoda w klasie typu **ManagedBean** do walidacji danych wprowadzanych za pomocą **h:inputText**

```
public void validateName(FacesContext context, UIComponent toValidate,
                         Object value)
  {  FacesMessage message;
     String input = (String) value;
     if (input.charAt(0)>96)                    //pierwsza litera powinna być duża
      { ((UIInput) toValidate).setValid(false);      //wynik negatywny walidacji
        message = new FacesMessage("Brak dużej litery na poczatku slowa");
        context.addMessage(toValidate.getClientId(context), message);    }
     }
```

Przykład 3 (cd) - Dodawanie atrybutu validator do znacznika input (rozbudowa przykładu 2 z wykładu 2) – kod Facelets strony index.xhtml

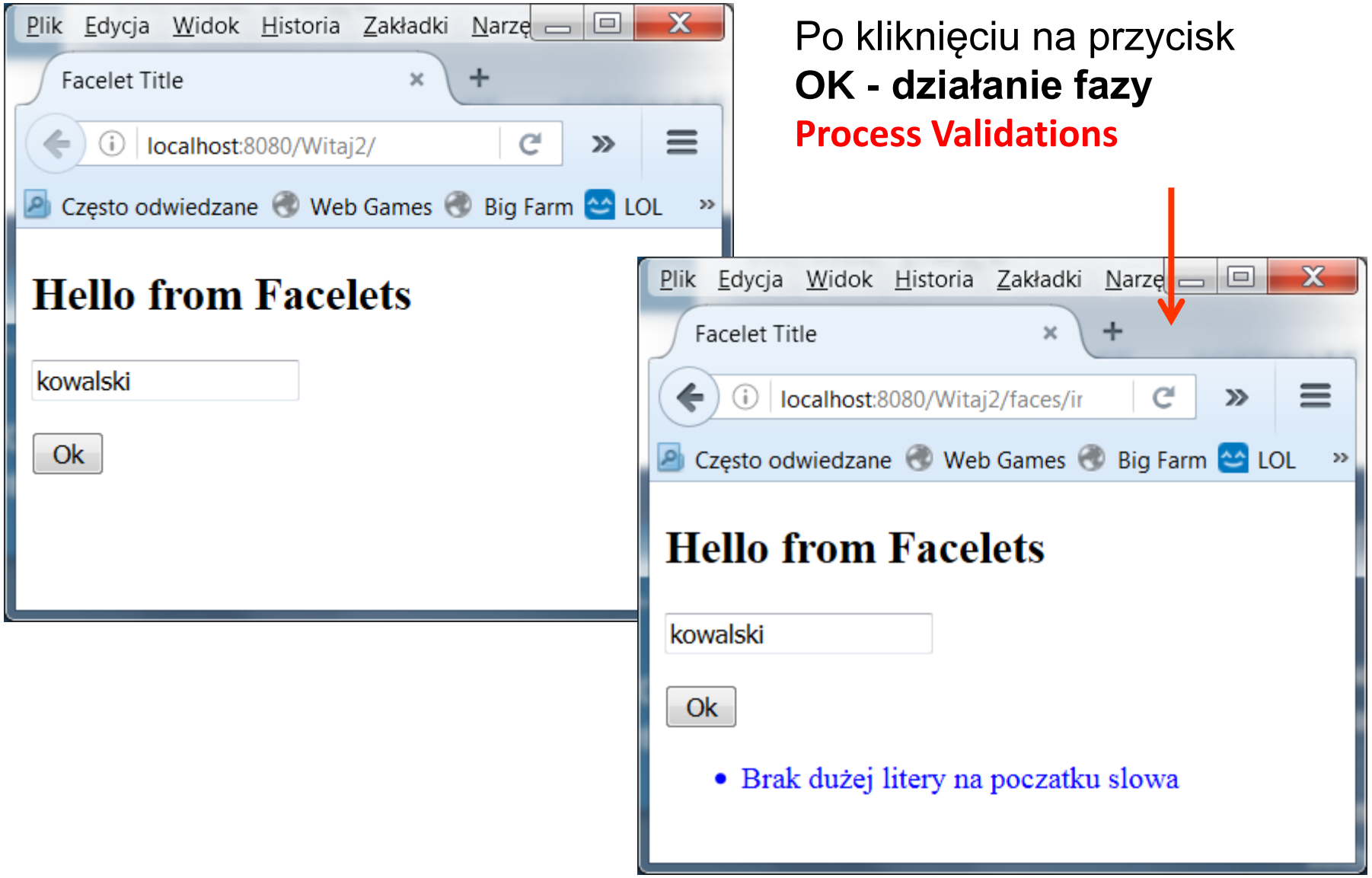
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <h2> Hello from Facelets </h2>
            <h:inputText
                id="name"
                title="Imie: "
                value="#{personalia.name}"
                required="true"
                requiredMessage="Blad: Podaj imie."
                maxlength="10"
                validator="#{personalia.validateName}"
                />
            <p> </p>
            <h:commandButton  id="ok" value="Ok" action="rezultat">
            </h:commandButton>
        </h:form>
    </h:body>
</html>
```

Metody w klasie typu **ManagedBean** do:

1) **setName -** przekazania danych wprowadzanych na stronie index.xhtml

2) **validateName -** walidacja danych wprowadzanych za pomocą **h:inputText**

```java
package warstwa_internetowa;
import java.io.Serializable;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
@ManagedBean
@SessionScoped
public class Personalia implements Serializable{
  private static final long serialVersionUID = 54433511513968687 24L;
  public Personalia() {    }
  private String name;
  public String getName() {return name;    }
  public void setName(String name) { this.name = name;    }
  public void validateName(FacesContext context, UIComponent toValidate, Object value)
    { FacesMessage message;
      String input = (String) value;
      if (input.charAt(0)>96)
        { ((UIInput) toValidate).setValid(false);  //wynik negatywny walidacji
          message = new FacesMessage("Brak dużej litery na poczatku slowa");
          context.addMessage(toValidate.getClientId(context), message);  }
    }
}
```



Po kliknięciu na przycisk
OK - działanie fazy
Process Validations

Przykład 4 - Dodawanie atrybutu valueChangeListener do znacznika input (rozbudowa przykładu 3) w celu zliczania zmian wartości wprowadzanej do pola name.

```
<h:inputText
    id="name"
    title="Imie: "
    value="#{personalia.name}"
    required="true"
    requiredMessage="Blad: Podaj imie."
    maxlength="10"
    validator="#{personalia.validateName}"
    valueChangeListener="#{personalia.processValueChange}">
</h:inputText>
```

Metoda w klasie typu Managed Bean do obsługi zdarzeń **Value-Change Event** generowanych przez widoki komponentów stron JSF

```
public void processValueChange(ValueChangeEvent event)
                                throws AbortProcessingException
 { if (null != event.getNewValue())
    { licznik++;}          //zliczanie zmian atrybutu name w czasie sesji
 }
```

Przykład 4 (cd) - Dodawanie atrybutu valueChangeListener do znacznika input (rozbudowa przykładu 3) w celu zliczania zmian wartości wprowadzanej do pola name - kod Facelets strony 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://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <h2> Hello from Facelets </h2>
            <h:inputText
                id="name"
                value="#{personalia.name}"
                required="true"
                requiredMessage="Blad: Podaj imie."
                validator="#{personalia.validateName}"
                valueChangeListener="#{personalia.processValueChange}">
            </h:inputText>
            <p>    </p>
            <h:commandButton  id="ok" value="Ok" action="rezultat">
            </h:commandButton>
        </h:form>
    </h:body>
</html>
```

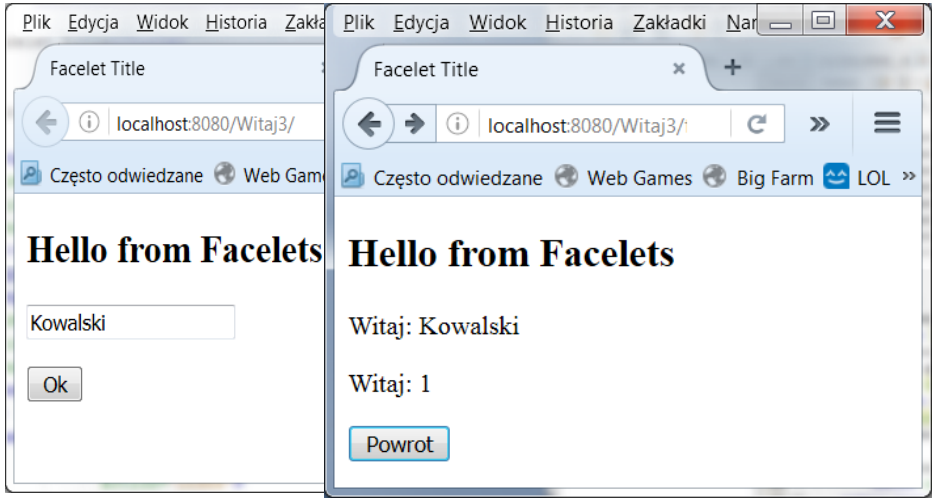
Metody w klasie typu **ManagedBean** do:
1)setName
2)validateName
3)processValueChange
obsługa zdarzenia związana ze zmianą wartości atrybutu **value**

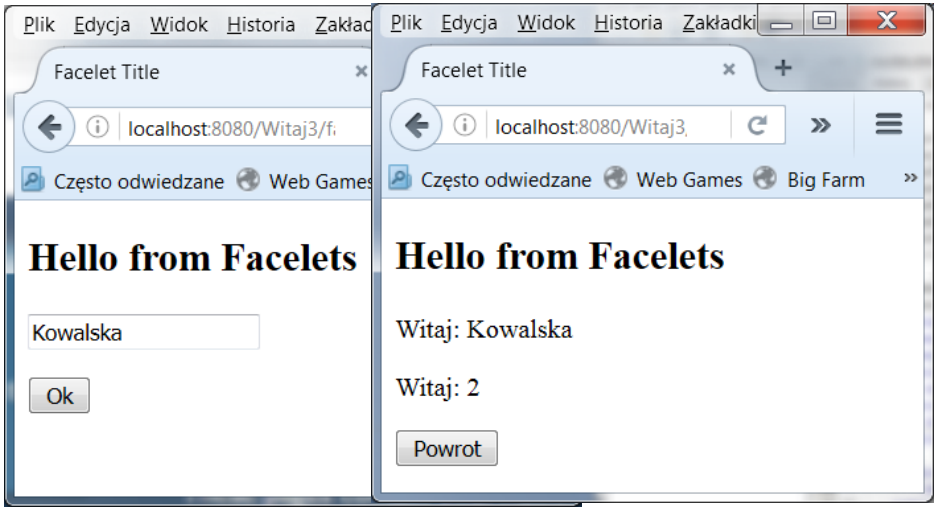
Przykład 4 (cd) – wynik obsługi zdarzenia.
Komponent typu Personalia musi mieć zakres SessionScoped

```
package warstwa_internetowa;
import java.io.Serializable;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ValueChangeEvent;
@ManagedBean
@SessionScoped
public class Personalia implements Serializable{
    private static final long serialVersionUID = 544335115139686872 4L;
    public int licznik;
    private String name;
 public int getLicznik()                  { return licznik; }
 public void setLicznik(int licznik)      { this.licznik = licznik;}
 public Personalia()                      { }
 public String getName()                  {return name; }
 public void setName(String name)         {this.name = name; }
 public void validateName(FacesContext context, UIComponent toValidate, Object value) {
        FacesMessage message;
        String input = (String) value;
        if (input.charAt(0) > 96) {
            ((UIInput) toValidate).setValid(false);  //wynik negatywny walidacji
            message = new FacesMessage("Brak dużej litery na poczatku slowa");
            context.addMessage(toValidate.getClientId(context), message);}
    }
public void processValueChange(ValueChangeEvent event) throws AbortProcessingException{
        if (null != event.getNewValue()) {
            licznik++;}
    }
}
```

Przykład 4 (cd) – wynik obsługi zdarzenia (strona rezultat.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://xmlns.jcp.org/jsf/html">
 <h:head>
   <title>Facelet Title</title>
 </h:head>
 <h:body>
   <h2> Hello from Facelets </h2>
   <h:form>
       <h:outputLabel
           id="rezultat"
           title="rezultat"
           value="Witaj: #{personalia.name}">
       </h:outputLabel>
       <p></p>
       <h:outputLabel
           id="rezultat1"
           title="rezultat1"
           value="Witaj: #{personalia.licznik}">
       </h:outputLabel>
     <p></p>
     <h:commandButton id="Powrot"
                      value="Powrot"
                      action="index">
     </h:commandButton>
   </h:form>
 </h:body>
</html>
```





Przykład 5 – inny sposób obsługi Value Change Listener

Przykłady znaczników typu input korzystających z obsługi zdarzeń

Zastosowanie zagnieżdżonego znacznika typu <f: valueChangeListener i atrybutu binding, który wskazuje na komponent implementujący metodę obsługi zdarzenia interfejsu ValueChangeListener

```
<h:inputText
    id="name"
    title="Imie: "
    value="#{personalia.name}"
    required="true"
    requiredMessage="Blad: Podaj imie."
    maxlength="10"
    validator="#{personalia.validateName}">
     <f:valueChangeListener binding="#{personalia}"/>
</h:inputText>
```

Zagnieżdżony znacznik **f:valueChangeListener**

Metoda w klasie typu Managed Bean **implementująca metodę interfejsu ValueChangeListener** do obsługi zdarzeń **Value-Change Event**

```
public void processValueChange(ValueChangeEvent event)
                                throws AbortProcessingException
 { if (null != event.getNewValue())
    { licznik++;}          //zliczanie zmian atrybutu name w czasie sesji
 }
```

Przykład 5 (cd) – wynik obsługi zdarzenia (strona 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://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
 <h:head>
   <title>Facelet Title</title>
 </h:head>
 <h:body>
   <h:form>
     <h2> Hello from Facelets </h2>
     <h:inputText
        id="name"
        title="Imie: "
        value="#{personalia.name}"
        required="true"
        requiredMessage="Blad: Podaj imie."
        maxlength="10"
        validator="#{personalia.validateName}">
         <f:valueChangeListener binding="#{personalia}"/>
     </h:inputText>
     <p>
     </p>
    <h:commandButton  id="ok" value="Ok" action="rezultat">
     </h:commandButton>
    </h:form>
 </h:body>
</html>
```

Metody w klasie typu **ManagedBean** do:

1)setName

2)validateName

3)processValueChange

obsługa zdarzenia związana ze zmianą wartości atrybutu **value. Atrybut binding wskazuje, gdzie znajduje się metoda do obsługi tego zdarzenia**

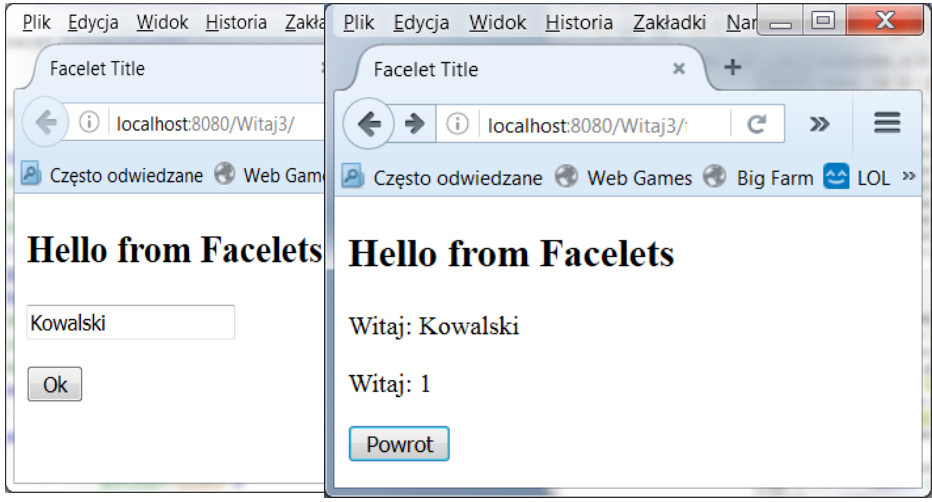
Przykład 5 (cd) – wynik obsługi zdarzenia

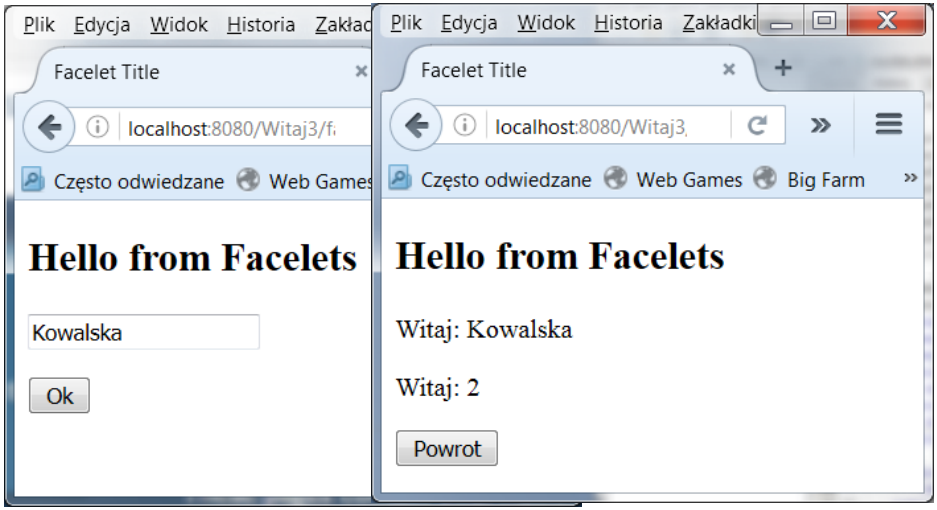
Komponent typu Personalia musi mieć zakres SessionScoped

```
package warstwa_internetowa;
import java.io.Serializable;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ValueChangeEvent;
import javax.faces.event.ValueChangeListener;
@ManagedBean
@SessionScoped
public class Personalia implements Serializable, ValueChangeListener{
    private static final long serialVersionUID = 544335115139686872 4L;
    public int licznik;
    private String name;
 public int getLicznik()                  { return licznik; }
 public void setLicznik(int licznik)      { this.licznik = licznik;}
 public Personalia()                      { }
 public String getName()                  {return name; }
 public void setName(String name)         {this.name = name; }
 public void validateName(FacesContext context, UIComponent toValidate, Object value) {
        FacesMessage message;
        String input = (String) value;
        if (input.charAt(0) > 96) {
            ((UIInput) toValidate).setValid(false);  //wynik negatywny walidacji
            message = new FacesMessage("Brak dużej litery na poczatku slowa");
            context.addMessage(toValidate.getClientId(context), message);}
    }
    @Override
    public void processValueChange(ValueChangeEvent event) throws AbortProcessingException{
        if (null != event.getNewValue()) {
            licznik++;}
    }
}
```

Przykład 5 (cd) – wynik obsługi zdarzenia (strona rezultat.xhtml taka sama jak w przykładzie 4)

```
<?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://xmlns.jcp.org/jsf/html">
 <h:head>
   <title>Facelet Title</title>
 </h:head>
 <h:body>
   <h2> Hello from Facelets </h2>
   <h:form>
       <h:outputLabel
           id="rezultat"
           title="rezultat"
           value="Witaj: #{personalia.name}">
       </h:outputLabel>
       <p></p>
       <h:outputLabel
           id="rezultat1"
           title="rezultat1"
           value="Witaj: #{personalia.licznik}">
       </h:outputLabel>
     <p></p>
     <h:commandButton id="Powrot"
                      value="Powrot"
                      action="index">
     </h:commandButton>
   </h:form>
 </h:body>
</html>
```





Przykład 6 – inny sposób obsługi Value-Change Listener

Przykłady znaczników typu input korzystających z obsługi zdarzeń

Zastosowanie zagnieżdżonego znacznika typu <f: valueChangeListener i atrybutu type, który wskazuje na definicję typu komponentu (wraz ze ścieżką pakietową) implementujący metodę obsługi zdarzenia interfejsu ValueChangeListener

```
<h:inputText
    id="name"
    title="Imie: "
    value="#{personalia.name}"
    required="true"
    requiredMessage="Blad: Podaj imie."
    maxlength="10"
    validator="#{personalia.validateName}">
     <f:valueChangeListener  type="warstwa_internetowa.nameChanged" />
</h:inputText>
```

Przykład 6 (cd) – definicja nowego słuchacza zdarzeń

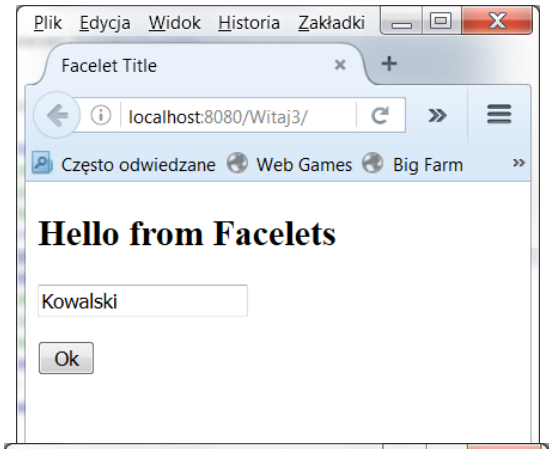
```
package warstwa_internetowa;

import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ValueChangeEvent;
import javax.faces.event.ValueChangeListener;

public class nameChanged implements ValueChangeListener {
 @Override
 public void processValueChange(ValueChangeEvent event)
                                 throws AbortProcessingException
 { if (null != event.getNewValue()) {
   FacesContext.getCurrentInstance().getExternalContext().
     getSessionMap().put("name", event.getNewValue()); }
    }
}
```

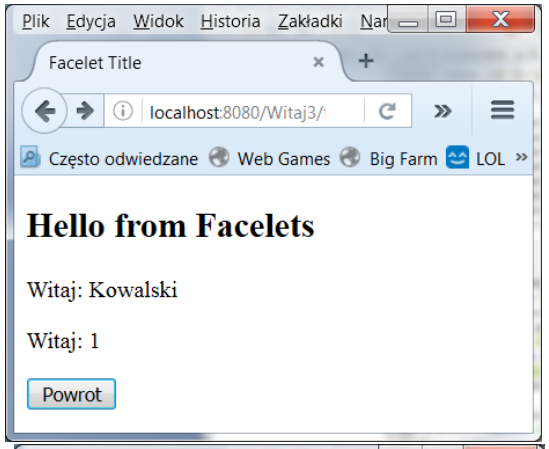
Przykład 6 (cd) – definicja metody getLicznik w klasie typu ManagedBean powiązanej z polem licznik znacznika outputLabel na stronie rezultat.xhtml

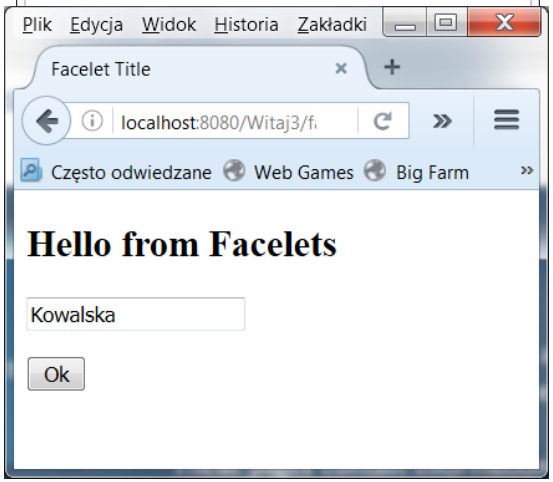
```
package warstwa_internetowa;
import java.util.Map;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
@ManagedBean
@SessionScoped
public class Personalia {
  private static final long serialVersionUID = 544335115139686872 4L;
  public int licznik=0;
  private String name;
  public int getLicznik()
    {Map<String,Object> mapa=FacesContext.getCurrentInstance().
                                getExternalContext().getSessionMap();
     String name1=(String)mapa.get("name");
     if(name.equals(name1)){
        mapa.remove("name");
        return ++licznik;}
     return licznik;
    }
  public void setLicznik(int licznik) {this.licznik = licznik; }
```

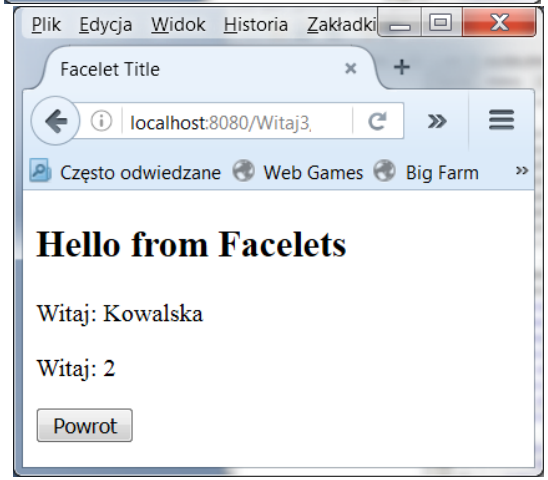
Komponent typu Personalia musi mieć zakres SessionScoped

```
public Personalia() {}
public String getName() { return name;}
public void setName(String name) {this.name = name;}
public void validateName(FacesContext context, UIComponent toValidate, Object value) {
        FacesMessage message;
        String input = (String) value;
        if (input.charAt(0) > 96) {
            ((UIInput) toValidate).setValid(false);  //wynik negatywny walidacji
            message = new FacesMessage("Brak dużej litery na poczatku slowa");
            context.addMessage(toValidate.getClientId(context), message);}
    }
}
```





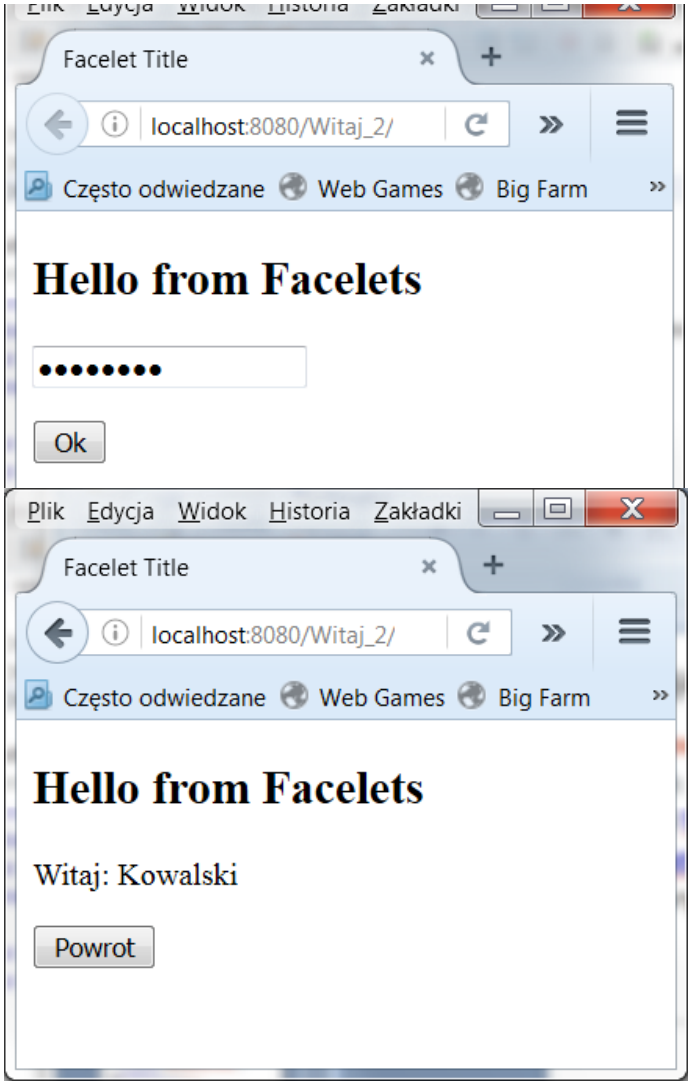




(4) Dodawanie wybranych znaczników – inputSecret
Przykład 7 (modyfikacja przykładu 2 z wykładu 2)

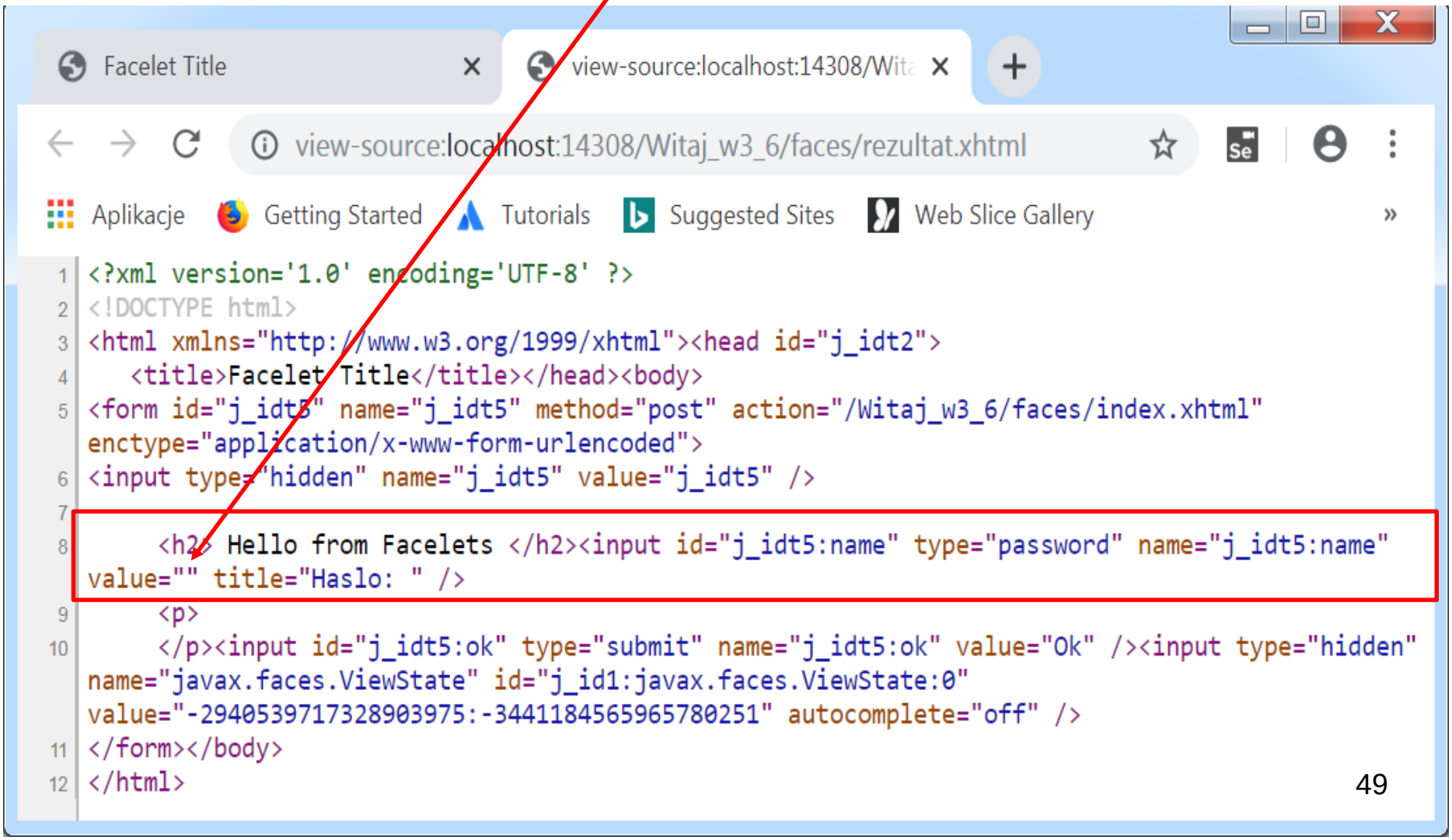
Zastosowanie znacznika h:inputSecret - znacznik umożliwia maskowanie wprowadzanych znaków. Wartość **false** atrybutu **redisplay** **pozwola uniknąć wyświetlenia tego ciągu znaków w zapytaniach lub w pliku HTML**

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Tr
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-tra
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
 <h:head>
   <title>Facelet Title</title>
 </h:head>
 <h:body>
   <h:form>
     <h2> Hello from Facelets </h2>
     <h:inputSecret
        id="name"
        redisplay="false"
        title="Haslo: "
        value="#{personalia.name}"/>
     <p>
     </p>
     <h:commandButton  id="ok" value="Ok"
                       action="rezultat">
     </h:commandButton>
    </h:form>
 </h:body>
</html>
```



(4) Dodawanie wybranych znaczników – inputSecret
Przykład 7 (cd) (modyfikacja przykładu 2 z wykładu 2)

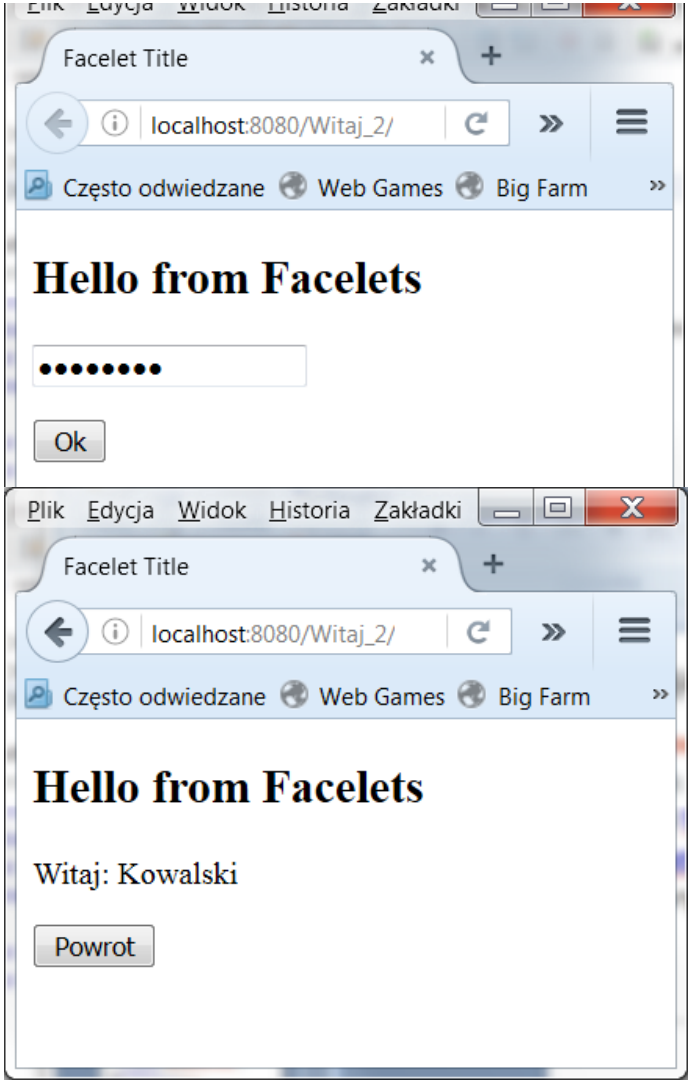
Zastosowanie znacznika h:inputSecret - znacznik umożliwia maskowanie wprowadzanych znaków. Wartość **false** atrybutu **redisplay** **pozwola uniknąć wyświetlenia tego ciągu znaków w zapytaniach lub w pliku HTML**



(4) Dodawanie wybranych znaczników – inputSecret
Przykład 7 (modyfikacja przykładu 2 z wykładu 2)

Zastosowanie znacznika h:inputSecret - znacznik umożliwia maskowanie wprowadzanych znaków. Wartość **true** atrybutu **redisplay** **pozwola na wyświetlenie tego ciągu znaków w zapytaniach lub w pliku HTML**

```
<?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://xmlns.jcp.org/jsf/html">
 <h:head>
   <title>Facelet Title</title>
 </h:head>
 <h:body>
   <h:form>
     <h2> Hello from Facelets </h2>
     <h:inputSecret
        id="name"
        redisplay="true"
        title="Haslo: "
        value="#{personalia.name}"/>
     <p>
     </p>
     <h:commandButton  id="ok" value="Ok"
                       action="rezultat">
     </h:commandButton>
    </h:form>
 </h:body>
</html>
```



(4) Dodawanie wybranych znaczników – inputSecret
Przykład 7 (cd) (modyfikacja przykładu 2 z wykładu 2)

Zastosowanie znacznika h:inputSecret - znacznik umożliwia maskowanie wprowadzanych znaków. Wartość **true** atrybutu **redisplay** **pozwola na wyświetlenie tego ciągu znaków w zapytaniach lub w pliku HTML**