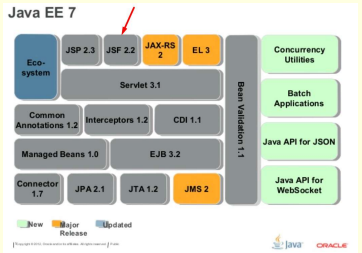
Wykład
JSF

JAVA II

Dr inż. Damian Raczyński

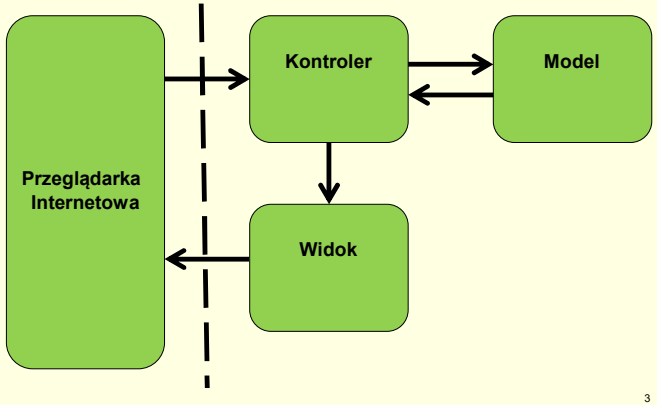
JSF

- Framework do budowania aplikacji WEB w Javie
- Oficjalny standard Java Enterprise Edition
- Bazuje na szablonie Model-Widok-Kontroler (MVC)

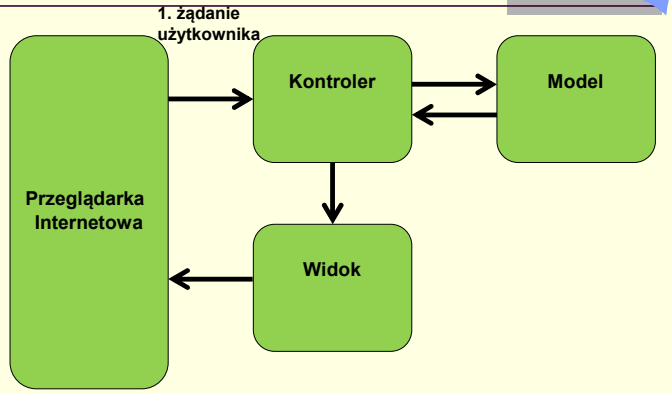


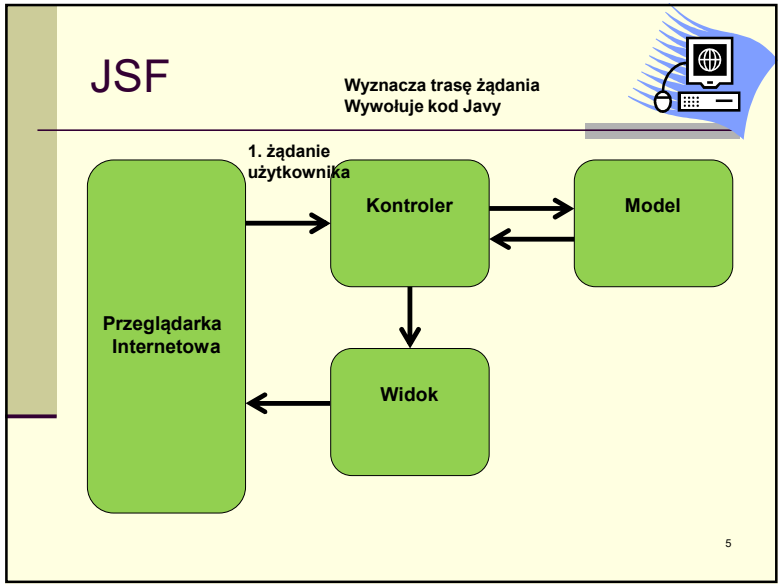
źródło: https://javastart.pl/baza-wiedzy/java-ee/czym-jest-java-ee

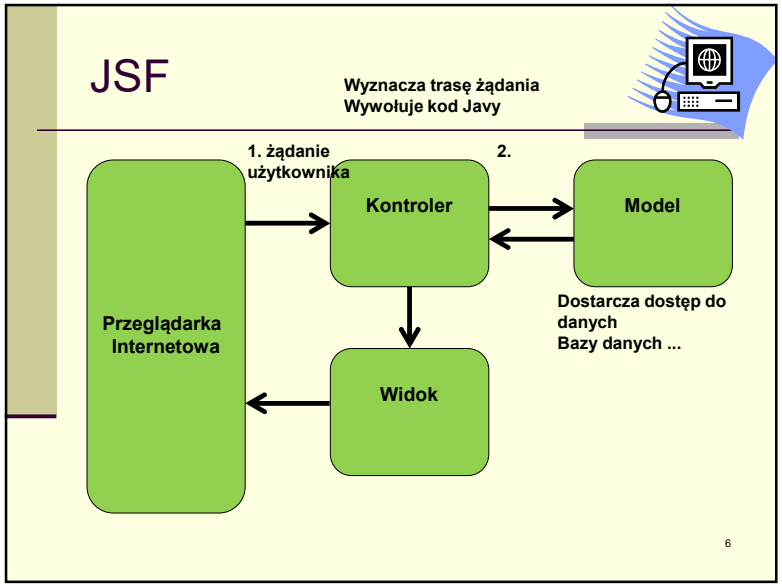
JSF

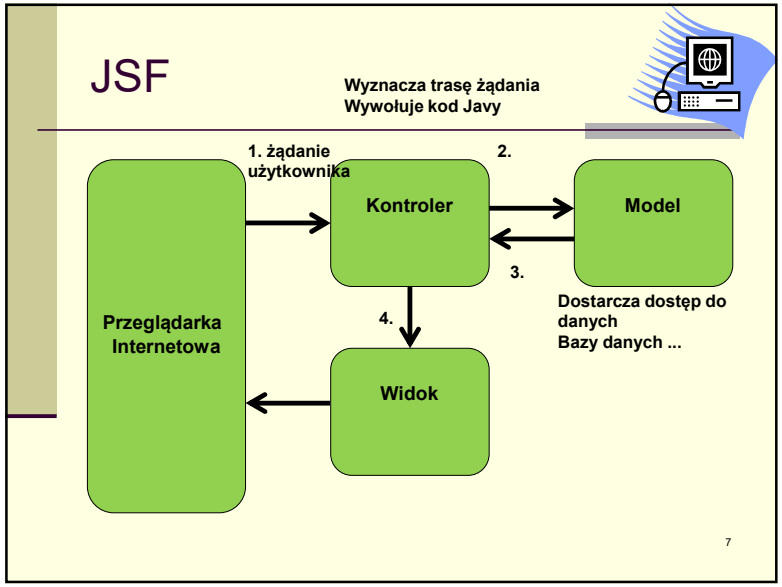


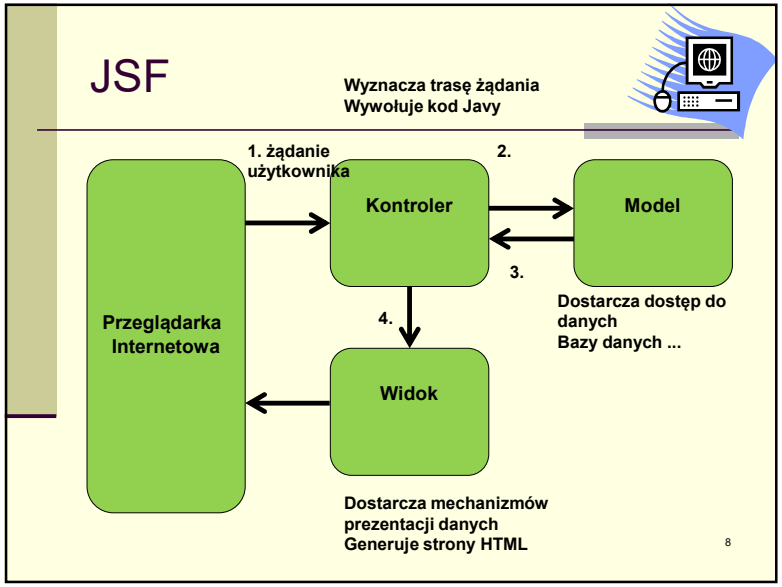
JSF




JSF
Wyznacza trasę żądania
Wywołuje kod Javy
1. żądanie użytkownika
Kontroler
Model
Przeglądarka Internetowa
Widok
5

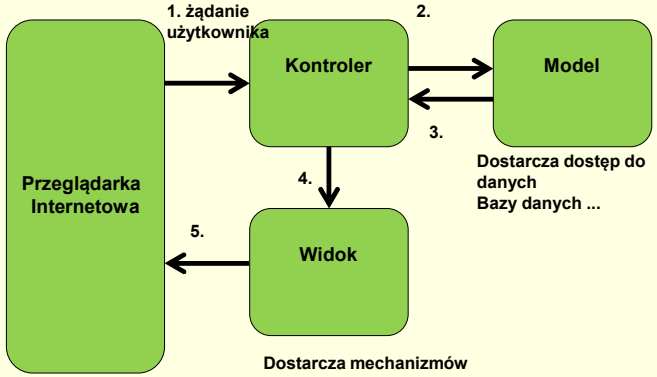

JSF
Wyznacza trasę żądania
Wywołuje kod Javy
1. żądanie użytkownika
2.
Kontroler
Model
Dostarcza dostęp do danych
Bazy danych ...
Przeglądarka Internetowa
Widok
6


JSF
Wyznacza trasę żądania
Wywołuje kod Javy
1. żądanie użytkownika
2.
Kontroler
Model
3.
4.
Dostarcza dostęp do danych
Bazy danych ...
Przeglądarka Internetowa
Widok
7

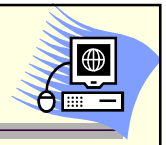

JSF
Wyznacza trasę żądania
Wywołuje kod Javy
1. żądanie użytkownika
2.
Kontroler
Model
3.
4.
Dostarcza dostęp do danych
Bazy danych ...
Przeglądarka Internetowa
Widok
Dostarcza mechanizmów prezentacji danych
Generuje strony HTML
8

JSF

Wyznacza trasę żądania
Wywołuje kod Javy



JSF



Zalety korzystania z JSF:

- STANDARDOWY sposób na budowę aplikacji WEB w Java
- Dostarcza zbioru przydatnych komponentów UI (wiele komponentów wizualnych)
- Pomaga w zarządzaniu stanem aplikacji
- Przetwarza dane: Walidacja, konwersja, ...
- Bardzo dużo dodatkowych komponentów – darmowych np. OpenFaces, RichFaces, PrimeFaces ...

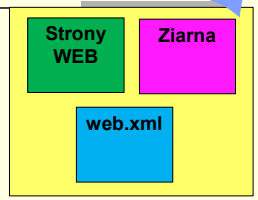




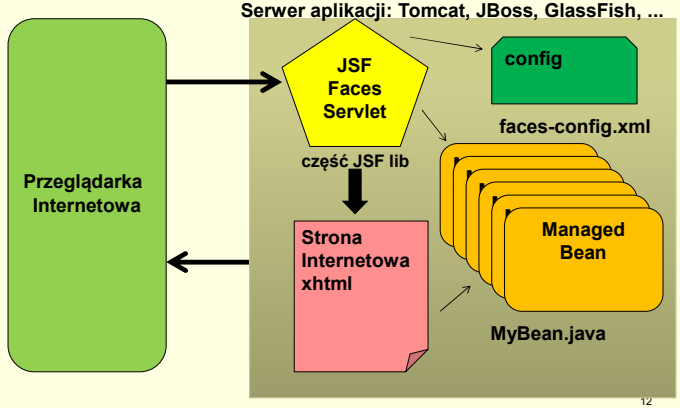
JSF

Komponenty aplikacji JSF:

- Zbiór stron WEB z komponentami UI
- Zbiór ziaren Java
- Deskryptor wdrożenia (web.xml)
- Opcjonalnie
plik konfiguracji aplikacji (faces-config.xml)
własne obiekty, komponenty, znaczniki, walidatory



JSF



JSF

Wersje JSF

Wersja JSF	Data premiery	wersja JEE
JSF 1.0	2004	J2EE 1.4
JSF 1.2	2006	Java EE 5
JSF 2.0	2009	Java EE 6
JSF 2.2	2013	Java EE 7
JSF 2.3	2017	Java EE 8

13

JSF

Prosty przykład:

index.xhtml

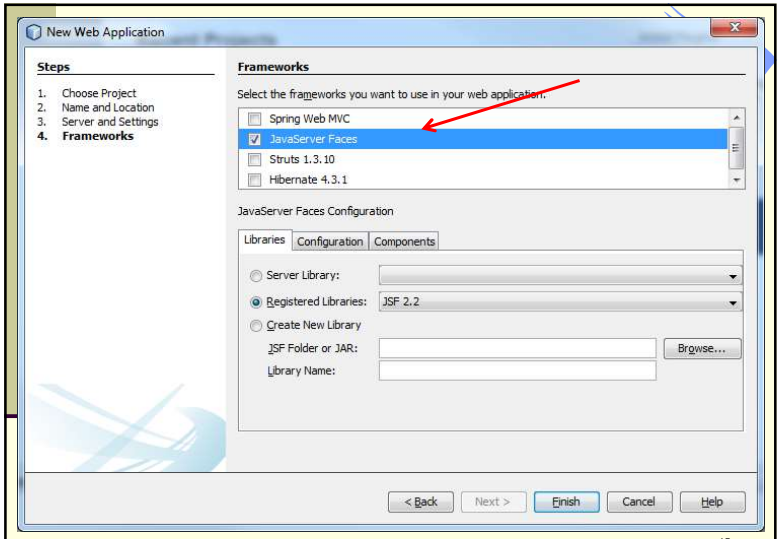
Jak masz na imię? Submit

response.xhtml

Witaj, Imię Nazwisko

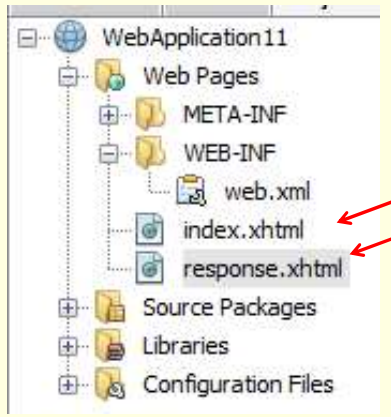
14

JSF



15

JSF



16

JSF

index.xhtml:

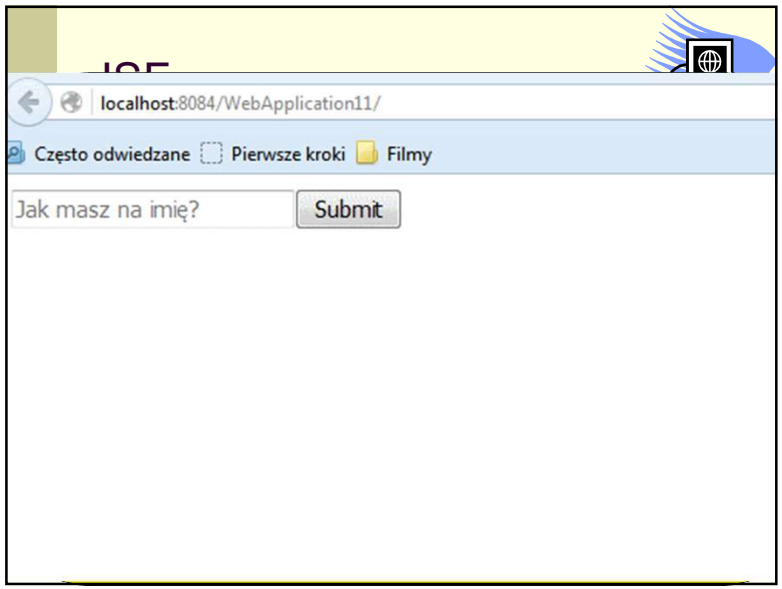
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:a="http://xmlns.jcp.org/jsf/passthrough">
    <h:head>
        <title>Prosty przykład</title>
    </h:head>
    <h:body>
        <h:form>
            <h:inputText id="imie" value="#{theUserName}"
                         a:placeholder="Jak masz na imię?"/>
            <h:commandButton value="Submit" action="response"
/>
        </h:form>
    </h:body>
</html>
```

JSF

response.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>Prosty przykład - odpowiedź</title>
    </h:head>
    <h:body>
        Witaj, #{theUserName}
    </h:body>
</html>
```

JSF



JSF

Wszystkie aplikacje JSF muszą w pliku web.xml mapować żądania użytkownika do kontrolera aplikacji (serwlet klasy `javax.faces.webapp.FacesServlet`):

web.xml:

```
<servlet>
    <display-name>FacesServlet</display-name>
    <servlet-name>FacesServlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>FacesServlet</servlet-name>
    <url-pattern> /prefix/* </url-pattern>
</servlet-mapping>
```

JSF

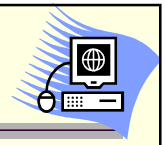
Konstrukcja #{ . . . } przybiera analogiczne znacznie jak konstrukcja ${ . . . } z języka EL.

Wyrażenie binduje wartość komponentu UI z właściwością/metodą ziarna (backing bean).

Czas określenia wartości wyrażenia w JSF jest ważny (ponieważ JSF posiada cykl życia podzielony na cały szereg faz - obsługa zdarzeń/konwersja danych/walidacja/…).

Użycie #{…} powoduje, że moment określania wartości wyrażenia jest określany przez JSF (wyrażenie oczekuje na aktualne dane).

JSF



Podstawowym pojęciem związanym z JSF jest komponent UI – element wchodzący w skład graficznych aplikacji www, mający swój stan i zachowanie.

Komponenty dziedziczą po klasie
`javax.faces.component.UIComponent`

W celu używania znaczników związanych z komponentami UI konieczne jest zastosowanie dyrektywy w jsp:

```
<%@ taglib uri="http://java.sun.com/jsf/html"
prefix="h" %>
```

lub dla strony xhtml:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
```

22

JSF

JSF zawiera szereg komponentów (generują kod HTML za programistę)

Przykłady:

JSF UI Komponent	Opis
h:form	kontener formularza
h:inputText	pole tekstowe
h:textArea	pole tekstowe wieloliniowe
h:selectBooleanCheckBox	przycisk opcji
h:selectOneRadio	przyciski radiowe
h:selectOneListBox	lista rozwijana
…	…

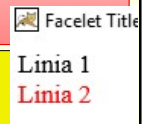
23

JSF

`h:outputText` - Komponent wyświetlający tekst. Tekst jest zawarty w atrybucie `value`. Wybrane atrybuty:

`id`	Identyfikator komponentu
`styleClass`	Nazwa klasy CSS
`value`	Wartość
`escape`	czy niebezpieczne znaki HTML/XML powinny zostać skonwertowane do postaci bezpiecznej

```
<h:body>
        <h:outputText value="Linia 1" escape="true"/>
        <br/>
        <h:outputText value="Linia 2" style='color:red;'/>
</h:body>
```



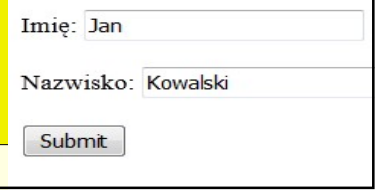
24

JSF

`h:inputText` – „input typu text". Wybrane atrybuty:

`id` Identyfikator komponentu
`value` wartość

```
<h:form>
Imię: <h:inputText id="imie" value="#{imie}"/>
<br/><br/>
Nazwisko: <h:inputText id="nazwisko" value="#{nazwisko}"/>
<br/><br/>
<h:commandButton value="Submit"/>
</h:form>
```



JSF

`h:outputLink` – odpowiednik <a ...> z HTML. Wybrane atrybuty:

`id` Identyfikator
`value` Wartość
`onmousemove` Kod JavaScript wykonywany dla ruchu myszą
`onmouseover` Javascript kod JavaScript, gdy mysz najedzie na komponent

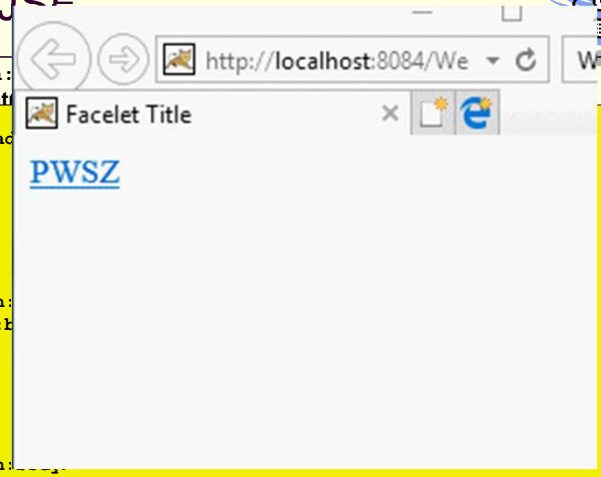
26

JSF

`h:outputLink` – odpowiednik <a ...> z HTML. Wybrane atrybuty:

```
<h:head>
        <title>Facelet Title</title>
        <script>
            function f(){
                alert('reakcja');
            }
        </script>
    </h:head>
    <h:body>
        <h:outputLink value="http://www.pwsz.nysa.pl"
                      onmouseover="f()"
                      id="link">
                          PWSZ
        </h:outputLink>
    </h:body>
```

JSF



JSF

JSF udostępnia 3 pola do wprowadzania tekstu:

`h:inputText`,
`h:inputSecret`,
`h:inputTextArea`

Znaczniki te są odpowiednikami znanymi z HTML'a znaczników:
`<input type="text" />`, `<input type="password" />`,
`<textarea> </textarea>`

Wybrane atrybuty:
`autocomplete` (`inputText` i `inputSecret`) – czy ma zostać użyta historia przeglądarki w celu autouzupełniania pól,
`maxlength` (`inputText`, `inputSecret`) – maksymalna długość tekstu

29

JSF

Wybrane atrybuty:

`readonly` (wszystkie) – określa czy jest dozwolone zaznaczanie i kopiowanie,

`disabled` – określa, czy pole jest włączone (czy można wpisywać tekst),

`redisplay` (`inputSecret`) – czy po przesłaniu formularza, wartość zawarta w polu powinna zostać przekazana z powrotem klientowi,

`cols`, `rows` (`inputTextArea`) – liczba kolumn i wierszy w komponencie,

`value` (wszystkie) – wartość komponentu.

JSF

`h:selectOneRatio` – przyciski radiowe (jeden w grupie może być zaznaczony)

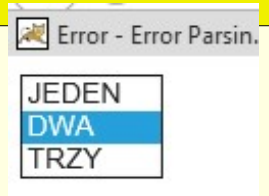
```
<h:form>
        <h:selectOneRadio id="radio" value="Wybor">
            <f:selectItem itemValue="1" itemLabel="JEDEN"/>
            <f:selectItem itemValue="2" itemLabel="DWA"/>
            <f:selectItem itemValue="3" itemLabel="TRZY"/>
        </h:selectOneRadio>
</h:form>
```

31

JSF

`h:selectOneListbox` -||- w odniesieniu do listy wyborów

```
<h:form>
        <h:selectOneListbox id="radio" value="Wybor">
            <f:selectItem itemValue="1" itemLabel="JEDEN"/>
            <f:selectItem itemValue="2" itemLabel="DWA"/>
            <f:selectItem itemValue="3" itemLabel="TRZY"/>
        </h:selectOneListbox>
</h:form>
```

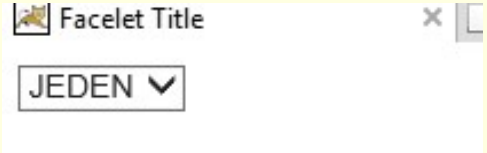


32

JSF

`h:selectOneMenu` -||- w odniesieniu do elementu rozwijanego

```
<h:form>
      <h:selectOneMenu id="radio" value="Wybor">
          <f:selectItem itemValue="1" itemLabel="JEDEN"/>
          <f:selectItem itemValue="2" itemLabel="DWA"/>
          <f:selectItem itemValue="3" itemLabel="TRZY"/>
      </h:selectOneMenu>
</h:form>
```



33

JSF

3 kontrolki umożliwiające dokonywania wielokrotnego wyboru:

`h:selectManyMenu`,
`h:selectManyListbox`,
`h:selectManyCheckbox`

Metody:
`public Object[] getSelectedValues()` – zwraca tablicę zawierającą zaznaczone elementy (lub null)

`public void setSelectedValues(Object[] tab)` – zaznacza elementy o wartościach przekazanych w tablicy.

34

JSF

Klasa `UIForm` i klasa potomna `HtmlForm` reprezentuje formularz zawarty na stronie Internetowej

Komponenty pochodne `UICommand` pozwalają na wywoływanie akcji (wysłanie formularza na serwer i wykonanie dodatkowych czynności).

Komponent przycisku – `h:commandButton` (klasa `HtmlCommandButton`),

`h:commandLink` (klasa `HtmlCommandLink`) – odsyłacz nie obsługuje formularzy w sposób naturalny (konieczne zastosowanie JavaScript).

35

JSF

Ziarna zarządzające (ang. Managed Beans)

- są zwykłymi klasami Java
- zazwyczaj używane do przechowywania danych z formularza
- mogą zawierać logikę biznesową
- tworzone i zarządzane przez JSF

Nie mylić z Enterprise Java Beans (EJB)

36

JSF

Konfiguracja ziaren:

- **W pliku XML (faces-config.xml):**

```
<managed-bean>
        <managed-bean-name>nazwa</managed-bean-name>
        <managed-bean-class>klasa</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

- **Konfiguracja ziarna z wykorzystaniem adnotacji:**

```
@ManagedBean(eager=true)
@ApplicationScoped
public class Nazwa{
. . .
}
```

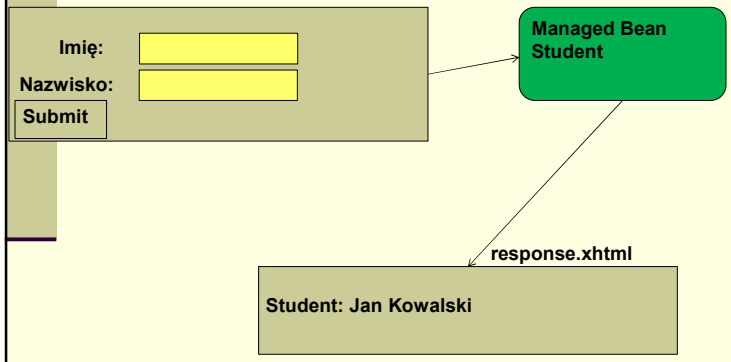
Adnotacja	odpowiednik XML
`@ApplicationScoped`	application
`@SessionScoped`	session
`@ViewScoped`	view (podczas interakcji użytkownika z pojedynczą stroną)
`@RequestScoped`	request

zarządzanie ziarnem jest "leniwe" (tworzone jest tylko gdy przyjdzie żądanie do aplikacji). Aby wymusić tworzenie ziarna zaraz po starcie aplikacji - `eager`

JSF

Prosty przykład:

index.xhtml



38

JSF

Wymagania dla ziaren zarządzających:

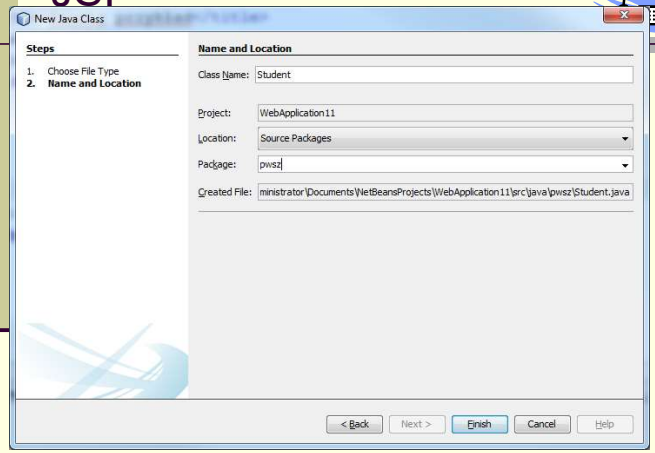
Ziarna zarządzające muszą spełniać następujące ograniczenia:

- **Muszą posiadać publiczny konstruktor bezparametrowy**
- **Udostępniać właściwości poprzez akcesory `get/set`**
- **JSF 2 dodał adnotację: `@ManagedBean`**

```
javax.faces.bean.ManagedBean
```

39

JSF



40

pwsz.Student.java:

```
package pwsz;
import javax.faces.bean.ManagedBean;     //2 możliwe opcje !
@ManagedBean
public class Student {
    public String getImie() {
        return imie;
    }
    public void setImie(String imie) {
        this.imie = imie;
    }
    public String getNazwisko() {
        return nazwisko;
    }
    public void setNazwisko(String nazwisko) {
        this.nazwisko = nazwisko;
    }
    private String imie;
    private String nazwisko;
    public Student(){
    }

}
```

JSF

JSF Expression Language

JSF expression language jest wykorzystywany w celu:
- **dostępu do właściwości managed bean**
- **wywołania innych funkcji związanych z logiką aplikacji**

Podstawowa składnia:
#{ <nazwaZiarna>.<nazwaWłaściwości> }

przykład:
#{student.imie} - odczyt/zapis właściwości

42

JSF

W celu odwołania do właściwości ziarna z JSF:

<h:inputText value=„#{student.imie}”/>

po Submit

Po przesłaniu formularza, JSF wywoła:
student.setImie(...)

43

JSF

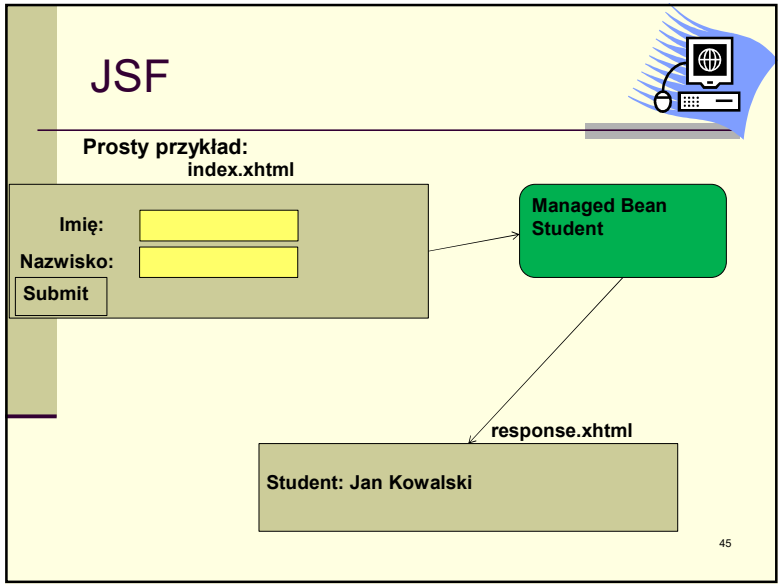
W celu odczytu właściwości ziarna postępujemy podobnie:

Student: #{student.imie}

W przypadku, gdy strona jest przetwarzana JSF wywoła:

student.getImie()

44



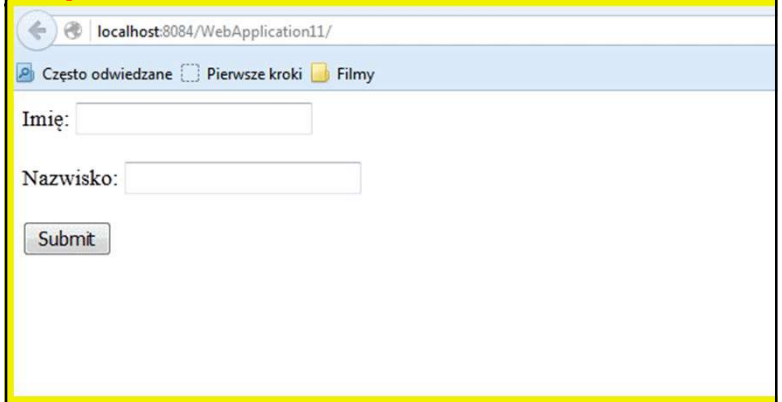
index.xhtml

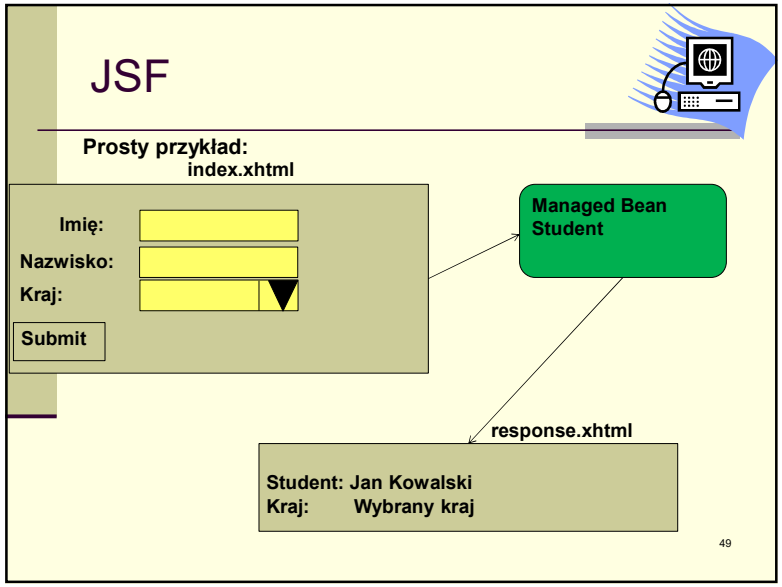
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:a="http://xmlns.jcp.org/jsf/passthrough">
    <h:head>
        <title>Rejestracja studenta</title>
    </h:head>
    <h:body>
        <h:form>
            Imię: <h:inputText id="imie"
value="#{student.imie}"/>
            <br/><br/>
            Nazwisko: <h:inputText id="nazwisko"
value="#{student.nazwisko}"/>
            <br/><br/>
            <h:commandButton value="Submit" action="response"/>
        </h:form>
    </h:body>
</html>
```

response.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>Powitanie studenta</title>
    </h:head>
    <h:body>
        Student: #{student.imie} #{student.nazwisko}
    </h:body>
</html>
```

response.xhtml



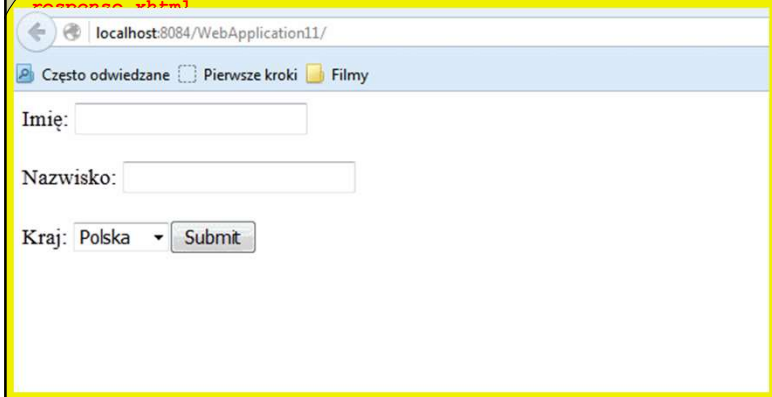



```
pwsz.Student.java:
@ManagedBean
public class Student {
    public String getKraj() {
        return kraj;    }
    public void setKraj(String kraj) {
        this.kraj = kraj;    }
    public String getImie() {
        return imie;    }
    public void setImie(String imie) {
        this.imie = imie;    }
    public String getNazwisko() {
        return nazwisko;    }
    public void setNazwisko(String nazwisko) {
        this.nazwisko = nazwisko;    }
    private String imie;
    private String nazwisko;
    private String kraj;
    public Student(){
    }
}
```

```
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:a="http://xmlns.jcp.org/jsf/passthrough"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head> <title>Rejestracja studenta</title>    </h:head>
    <h:body>        <h:form>
     Imię: <h:inputText id="imie" value="#{student.imie}"/>
           <br/><br/>
Nazwisko: <h:inputText id="nazwisko" value="#{student.nazwisko}"/>
           <br/><br/>
           Kraj:
           <h:selectOneMenu value="#{student.kraj}">
               <f:selectItem itemValue="Polska" itemLabel="Polska"/>
               <f:selectItem itemValue="Niemcy" itemLabel="Niemcy"/>
               <f:selectItem itemValue="Turcja" itemLabel="Turcja"/>
               <f:selectItem itemValue="Anglia" itemLabel="Anglia"/>
           </h:selectOneMenu>
           <h:commandButton value="Submit" action="response"/>
       </h:form>
    </h:body>
</html>
```

```
response.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>Powitanie studenta</title>
    </h:head>
    <h:body>
        Student: #{student.imie} #{student.nazwisko}
        <br/>
        Kraj pochodzenia: #{student.kraj}
    </h:body>
</html>
```

response.xhtml



JSF

Czy można lepiej ???

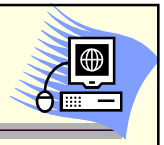
pwsz.Student.java:

```
@ManagedBean
public class Student {
    public List<String> getKrajOpcje() {
        return krajOpcje;
    }
    private List<String> krajOpcje;

            ...

public Student(){
        krajOpcje= new ArrayList<>();
        krajOpcje.add("Polska");
        krajOpcje.add("Niemcy");
        krajOpcje.add("Turcja");
        krajOpcje.add("Anglia");
    }
}
```

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:a="http://xmlns.jcp.org/jsf/passthrough"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <title>Rejestracja studenta</title>
    </h:head>
    <h:body>
        <h:form>
            Imię: <h:inputText id="imie" value="#{student.imie}"/>
            <br/><br/>
            Nazwisko: <h:inputText id="nazwisko"
value="#{student.nazwisko}"/>
            <br/><br/>
            Kraj:
            <h:selectOneMenu value="#{student.kraj}">
                <f:selectItems value="#{student.krajOpcje}"/>
            </h:selectOneMenu>
            <h:commandButton value="Submit" action="response"/>
        </h:form>
    </h:body>
</html>
```

JSF



Prosty przykład:

index.xhtml

Imię:

Nazwisko:

Hobby: ○ Java ○ C# ○ PHP

Submit

Managed Bean
Student

response.xhtml

Student: Jan Kowalski
Hobby: Wybrany Język

57

pwsz.Student.java:

```
@ManagedBean
public class Student {
    public String getUlubionyJezyk() {
        return ulubionyJezyk;    }
    public void setUlubionyJezyk(String ulubionyJezyk) {
        this.ulubionyJezyk = ulubionyJezyk;    }
    public String getImie() {
        return imie;    }
    public void setImie(String imie) {
        this.imie = imie;    }
    public String getNazwisko() {
        return nazwisko;    }
    public void setNazwisko(String nazwisko) {
        this.nazwisko = nazwisko;    }
    private String imie;
    private String nazwisko;
    private String ulubionyJezyk;
    public Student(){

    }
}
```

index.xhtml

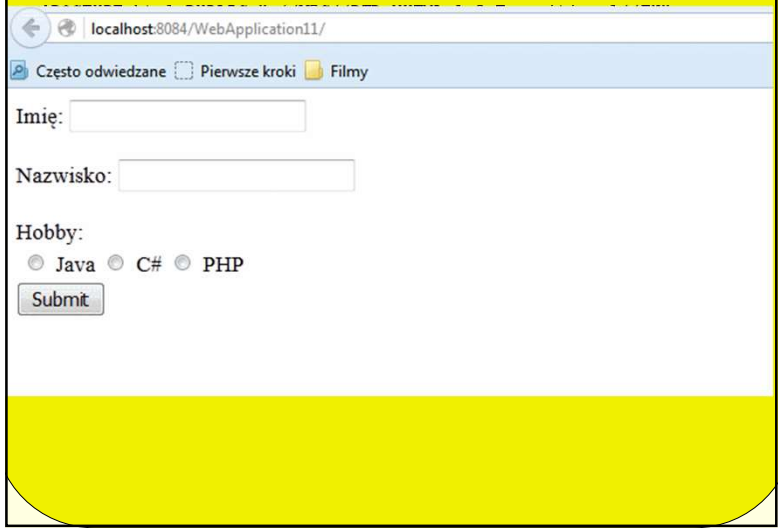
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:a="http://xmlns.jcp.org/jsf/passthrough"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <title>Rejestracja studenta</title>
    </h:head>
    <h:body>
        <h:form>
            Imię: <h:inputText id="imie" value="#{student.imie}"/>
            <br/><br/>
  Nazwisko: <h:inputText id="nazwisko" value="#{student.nazwisko}"/>
            <br/><br/>
            Hobby:
            <h:selectOneRadio value="#{student.ulubionyJezyk}">
                <f:selectItem itemValue="Java" itemLabel="Java"/>
                <f:selectItem itemValue="C#" itemLabel="C#"/>
                <f:selectItem itemValue="PHP" itemLabel="PHP"/>
            </h:selectOneRadio>
            <h:commandButton value="Submit" action="response"/>
        </h:form>
    </h:body>
</html>
```

response.xhtml

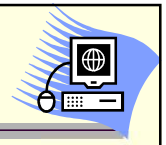
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Powitanie studenta</title>
    </h:head>
    <h:body>
        Student: #{student.imie} #{student.nazwisko}
        <br/>
        Hobby: #{student.ulubionyJezyk}
    </h:body>
</html>
```

response.xhtml

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



JSF



Tag UI:

JSP:

```
<%@ taglib prefix="ui"
    uri="http://java.sun.com/jsf/facelets" %>
```

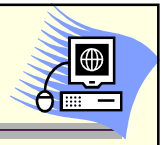
XML:

```
<anyxmlelement
    xmlns:ui="http://java.sun.com/jsf/facelets" />
```

Biblioteka znaczników dostarczająca funkcjonalności tworzenia szablonów stron (między innymi wstawianie sekcji, realizacja operacji w pętli …):

```
<ui:repeat value="#{obiekt.kolekcja}" var="licznik">
    . . .
</ui:repeat>
```

JSF



Tag F:

JSP:

```
<%@ taglib prefix="f"
    uri="http://xmlns.jcp.org/jsf/core" %>
```

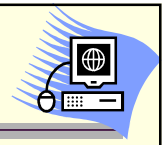
XML:

```
<anyxmlelement
    xmlns:f="http://xmlns.jcp.org/jsf/core" />
```

Biblioteka zawiera akcje, które są niezależne od sposobu renderowania elememementów.

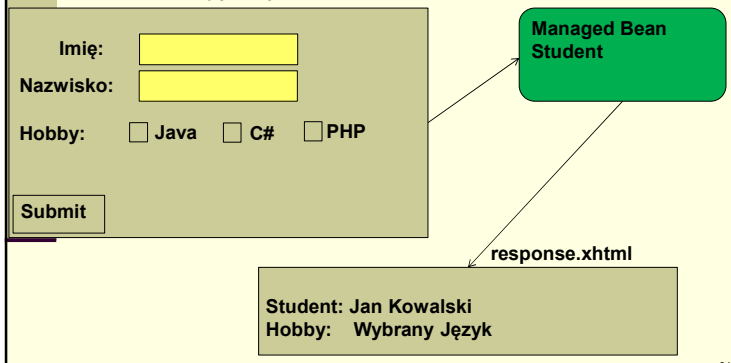
63

JSF



Prosty przykład:

index.xhtml



64

pwsz.Student.java:

```
@ManagedBean
public class Student {
    public String[] getUlubionyJezyk() {
        return ulubionyJezyk;    }
    public void setUlubionyJezyk(String[] ulubionyJezyk) {
        this.ulubionyJezyk = ulubionyJezyk;    }
    public String getImie() {
        return imie;    }
    public void setImie(String imie) {
        this.imie = imie;    }
    public String getNazwisko() {
        return nazwisko;    }
    public void setNazwisko(String nazwisko) {
        this.nazwisko = nazwisko;    }
    private String imie;
    private String nazwisko;
    private String [] ulubionyJezyk;
    public Student(){

    }
}
```

index.xhtml

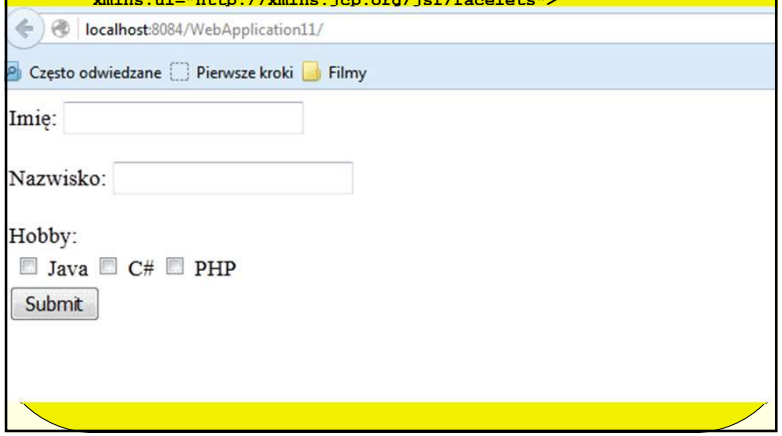
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:a="http://xmlns.jcp.org/jsf/passthrough"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <title>Rejestracja studenta</title>
    </h:head>
    <h:body>
        <h:form>
            Imię: <h:inputText id="imie" value="#{student.imie}"/>
            <br/><br/>
            Nazwisko: <h:inputText id="nazwisko"
value="#{student.nazwisko}"/>
            <br/><br/>
            Hobby:
            <h:selectManyCheckbox value="#{student.ulubionyJezyk}">
                <f:selectItem itemValue="Java" itemLabel="Java"/>
                <f:selectItem itemValue="C#" itemLabel="C#"/>
                <f:selectItem itemValue="PHP" itemLabel="PHP"/>
            </h:selectManyCheckbox>
            <h:commandButton value="Submit" action="response"/>
        </h:form>
    </h:body>
```

response.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:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Powitanie studenta</title>
    </h:head>
    <h:body>
        Student: #{student.imie} #{student.nazwisko}
        <br/>
        Hobby:
        <ul>
            <ui:repeat value="#{student.ulubionyJezyk}" var="temp">
                <li>#{temp}</li>
            </ui:repeat>
        </ul>
    </h:body>
</html>
```

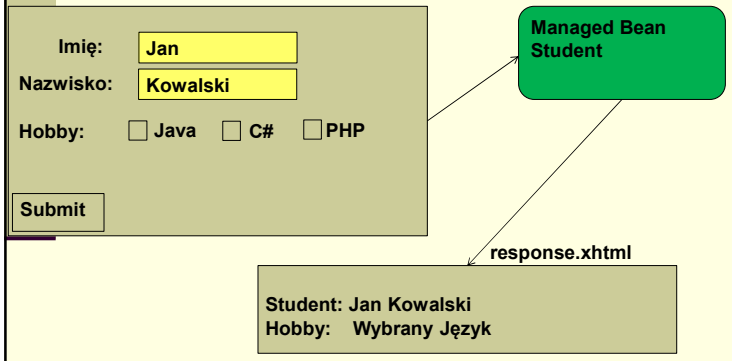
response.xhtml

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets">
```



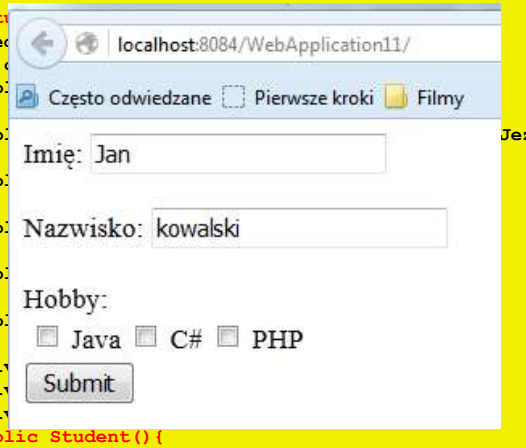
JSF

Prosty przykład:
index.xhtml



69

```
pwsz.Student.java:
@ManagedBean
public class Student {
    public String[] getUlubionyJezyk() {
        return ulubionyJezyk;    }
    public void setUlubionyJezyk(String[] ulubionyJezyk) {
        this.ulubionyJezyk = ulubionyJezyk;    }
    public String getImie() {
        return imie;    }
    public void setImie(String imie) {
        this.imie = imie;    }
    public String getNazwisko() {
        return nazwisko;    }
    public void setNazwisko(String nazwisko) {
        this.nazwisko = nazwisko;    }
    private String imie;
    private String nazwisko;
    private String [] ulubionyJezyk;
    public Student(){
        imie="Jan";
        nazwisko="kowalski";

    }
```




```
public Student(){
    imie="Jan";
    nazwisko="kowalski";

}
```

JSF

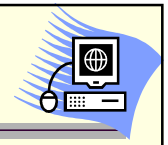
Komponenty `UI` pobierają wartości z klas skojarzonych (interakcja elementów wizualnych i elementów zawartych w klasach).

Rzutowanie między typami (np. `int` → `String`) zachodzi automatycznie (możliwe stworzenie klasy konwertera – o tym później).

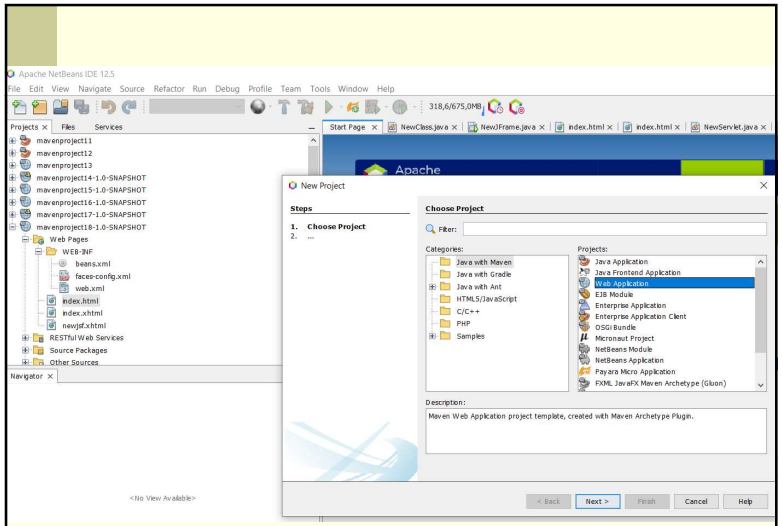
Wyrażenia `JSF EL` wykonują się dwustronnie (zapisują, kiedy jest wysyłany formularz, odczytują, gdy jest wyświetlany)

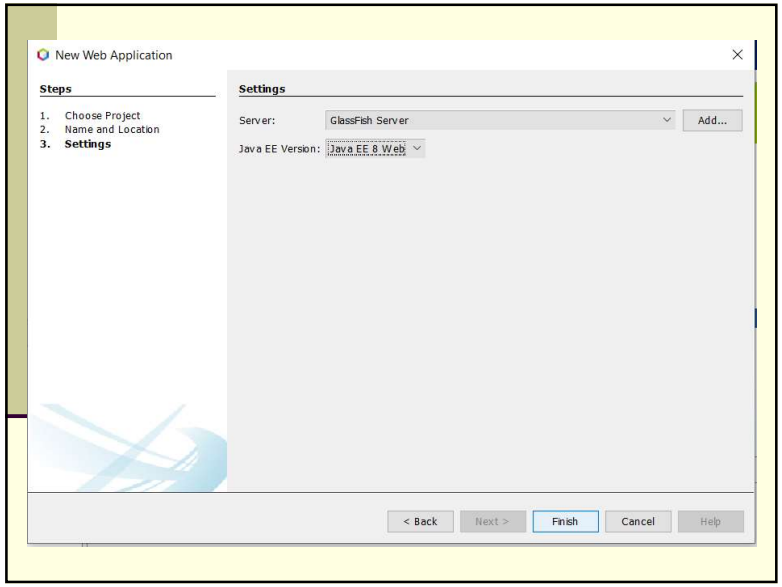
72

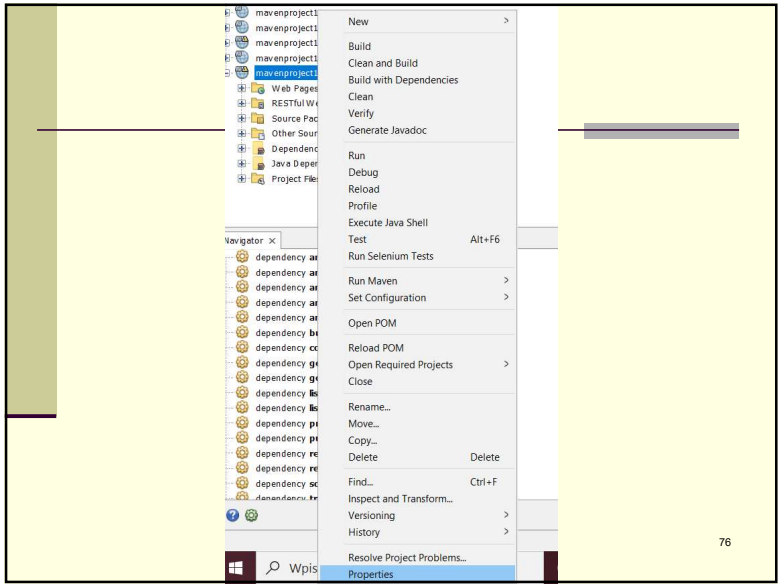
JSF - kalkulator

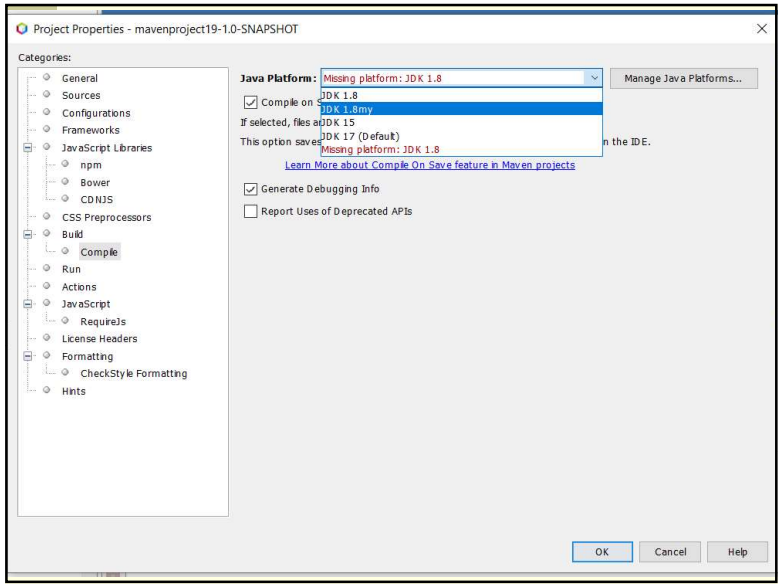


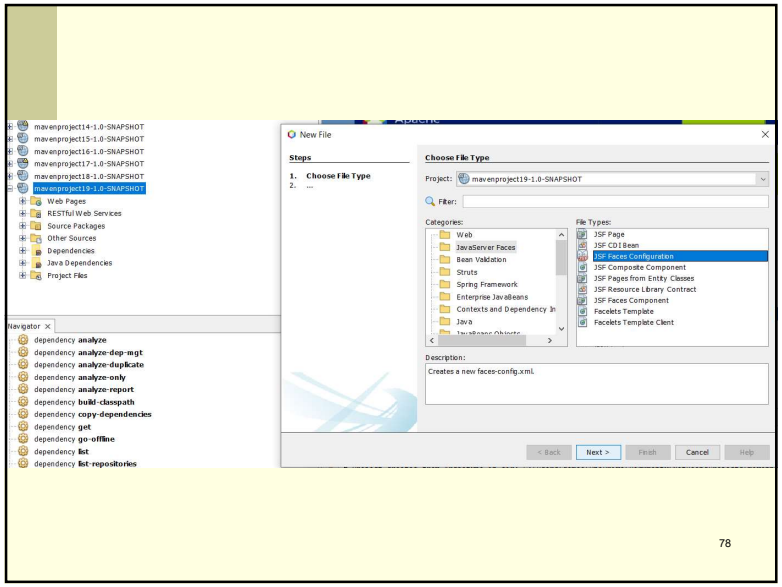
Przykład w NetBeans



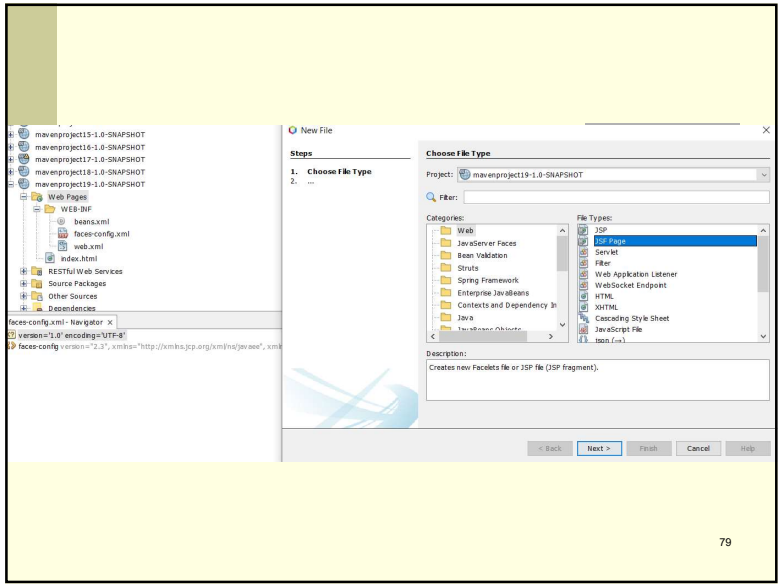




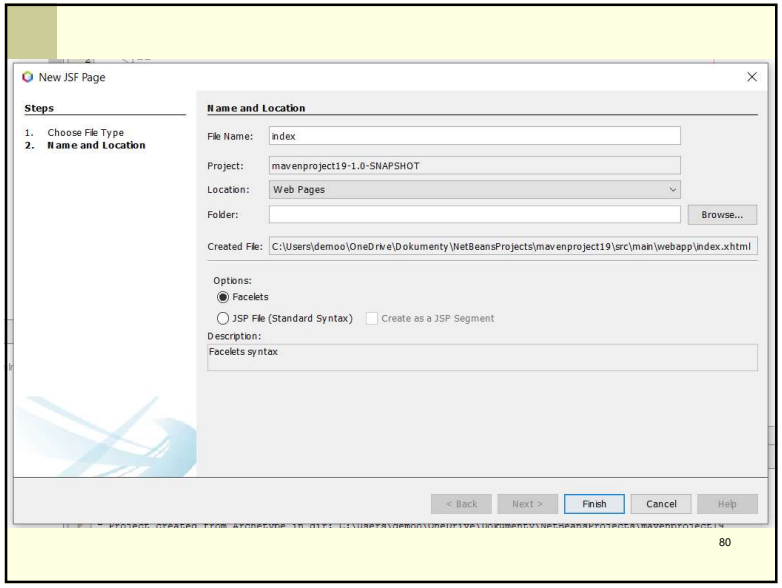

Project Properties - mavenproject19-1.0-SNAPSHOT
Categories:
General
Sources
Configurations
Frameworks
JavaScript Libraries
npm
Bower
CDNJS
CSS Preprocessors
Build
Compile
Run
Actions
JavaScript
RequireJs
License Headers
Formatting
CheckStyle Formatting
Hints
Java Platform:
Missing platform: JDK 1.8
Manage Java Platforms...
JDK 1.8
JDK 1.8my
JDK 15
JDK 17 (Default)
Missing platform: JDK 1.8
Compile on S
If selected, files a
This option saves
n the IDE.
Learn More about Compile On Save feature in Maven projects
Generate Debugging Info
Report Uses of Deprecated APIs
OK
Cancel
Help


New File
Steps
1. Choose File Type
2. ...
Choose File Type
Project:
mavenproject19-1.0-SNAPSHOT
Filter:
Categories:
Web
JavaServer Faces
Bean Validation
Struts
Spring Framework
Enterprise JavaBeans
Contexts and Dependency In
Java
File Types:
JSF Page
JSF CDI Bean
JSF Faces Configuration
JSF Composite Component
JSF Pages from Entity Classes
JSF Resource Library Contract
JSF Faces Component
Facelets Template
Facelets Template Client
Description:
Creates a new faces-config.xml
< Back
Next >
Finish
Cancel
Help
Navigator
dependency analyze
dependency analyze-dep-mgt
dependency analyze-duplicate
dependency analyze-only
dependency analyze-report
dependency build-classpath
dependency copy-dependencies
dependency get
dependency go-offline
dependency list
dependency list-repositories

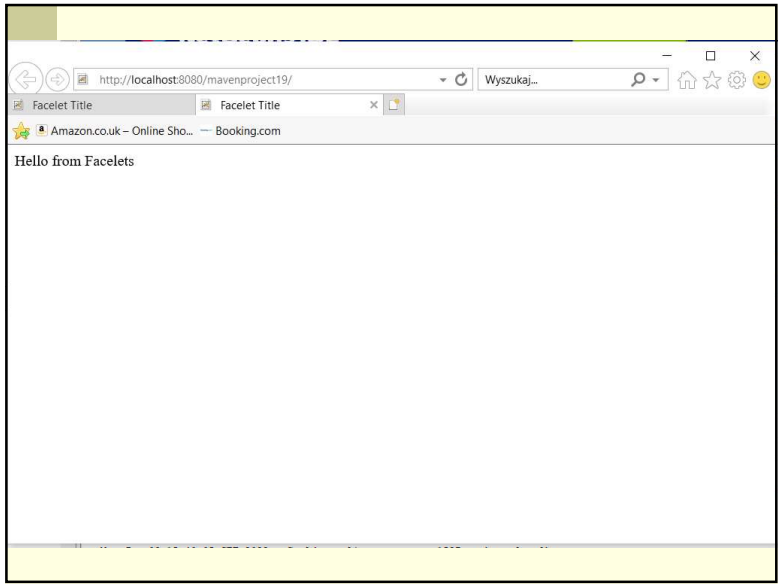
78


New File
Steps
1. Choose File Type
2. ...
Choose File Type
Project:
mavenproject19-1.0-SNAPSHOT
Filter:
Categories:
Web
JavaServer Faces
Bean Validation
Struts
Spring Framework
Enterprise JavaBeans
Contexts and Dependency In
Java
File Types:
JSP
JSF Page
Servlet
Filter
Web Application Listener
WebSocket Endpoint
HTML
XHTML
Cascading Style Sheet
JavaScript File
Description:
Creates new Facelets file or JSP file (JSP fragment).
< Back
Next >
Finish
Cancel
Help
mavenproject15-1.0-SNAPSHOT
mavenproject16-1.0-SNAPSHOT
mavenproject17-1.0-SNAPSHOT
mavenproject18-1.0-SNAPSHOT
mavenproject19-1.0-SNAPSHOT
Web Pages
WEB-INF
beans.xml
faces-config.xml
web.xml
index.html
RESTful Web Services
Source Packages
Other Sources
Dependencies
faces-config.xml - Navigator
version='1.0' encoding='UTF-8'
faces-config

79


New JSF Page
Steps
1. Choose File Type
2. Name and Location
Name and Location
File Name:
index
Project:
mavenproject19-1.0-SNAPSHOT
Location:
Web Pages
Folder:
Browse...
Created File:
C:\Users\demoo\OneDrive\Dokumenty\NetBeansProjects\mavenproject19\src\main\webapp\index.xhtml
Options:
Facelets
JSP File (Standard Syntax)
Create as a JSP Segment
Description:
Facelets syntax
< Back
Next >
Finish
Cancel
Help

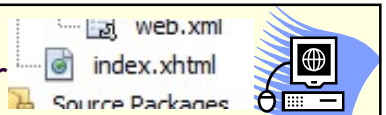
80




```
package kalkulator;
public class calc {
    private int wartosc1;
    private int wartosc2;
    private int wynik;
    public int getWartosc1(){        return wartosc1;    }
    public void setWartosc1(int wartosc1){
        this.wartosc1=wartosc1;
    }
    public int getWartosc2(){        return wartosc2;    }
    public void setWartosc2(int wartosc2){
        this.wartosc2=wartosc2;
    }
    public int getWynik(){        return wynik;    }
    public String licz(){
        this.wynik=this.getWartosc1()+this.getWartosc2();
        return null;
    }
}
```

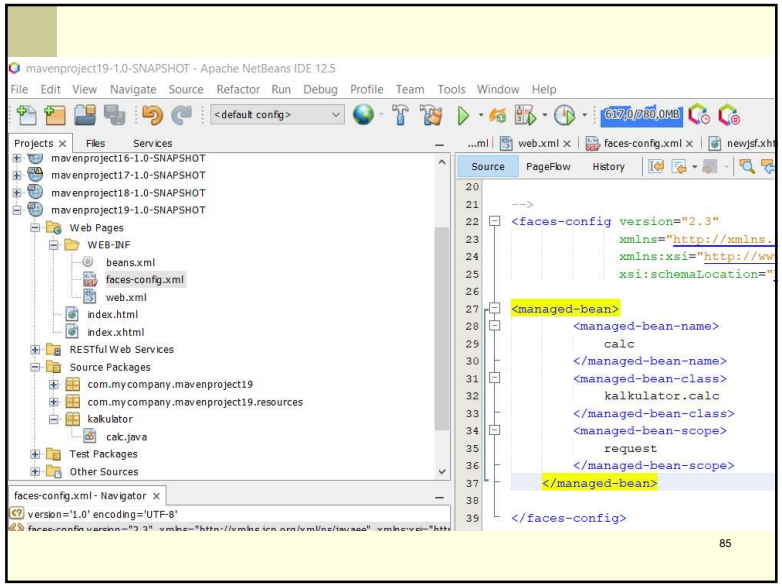
Source Packages
kalkulator
calc.java

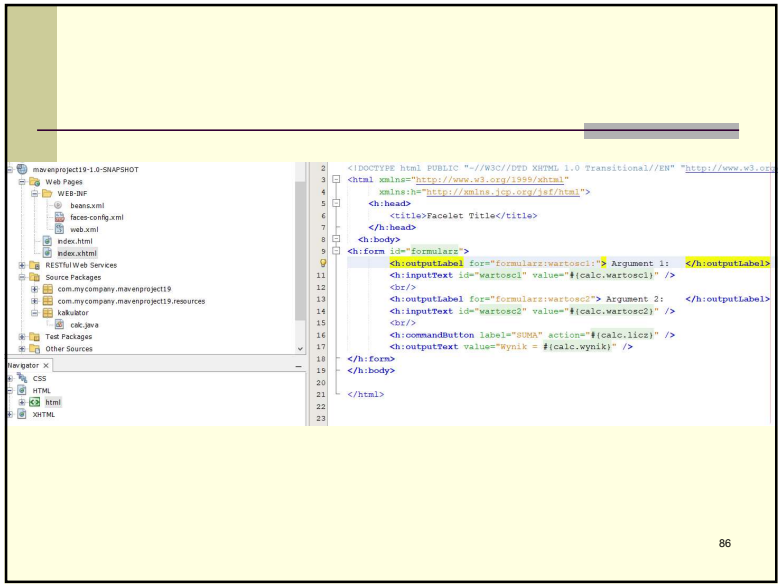
JSF - kalkulator




```
<h:body>
<h:form id="formularz">
      <h:outputLabel for="formularz:wartosc1:"> Argument 1:
      </h:outputLabel>
      <h:inputText id="wartosc1" value="#{calc.wartosc1}" />
      <br/>
      <h:outputLabel for="formularz:wartosc2"> Argument 2:
      </h:outputLabel>
      <h:inputText id="wartosc2" value="#{calc.wartosc2}" />
      <br/>
      <h:commandButton label="SUMA" action="#{calc.licz}" />
      <h:outputText value="Wynik = #{calc.wynik}" />
</h:form>
</h:body>
```

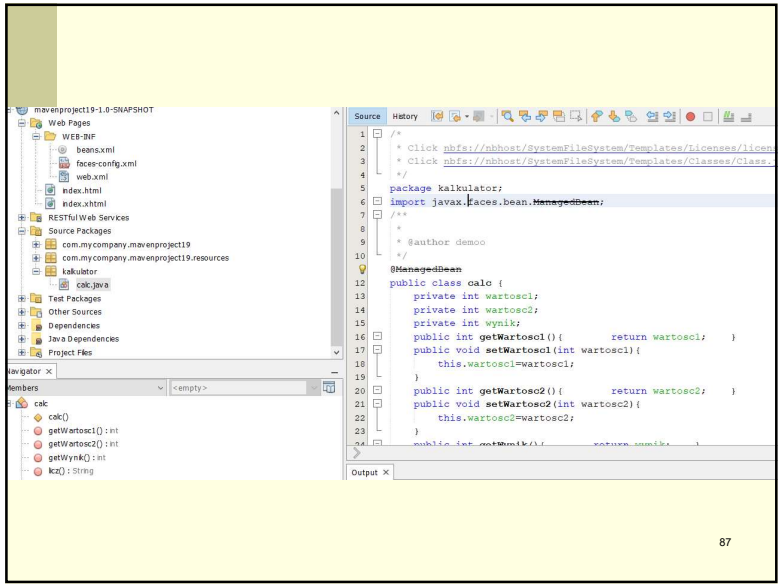
```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
              xmlns="http://xmlns.jcp.org/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
    <managed-bean>
        <managed-bean-name>
            calc
        </managed-bean-name>
        <managed-bean-class>
            kalkulator.calc
        </managed-bean-class>
        <managed-bean-scope>
            request
        </managed-bean-scope>
    </managed-bean>
</faces-config>
```

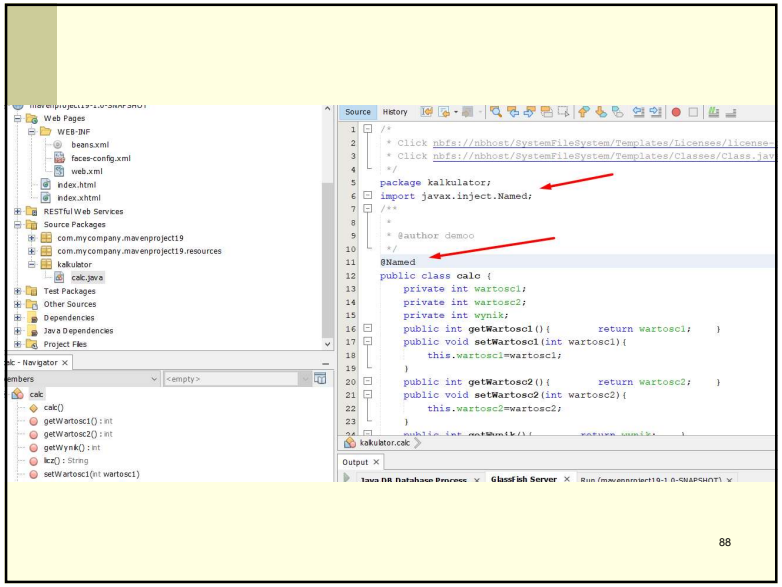

mavenproject19-1.0-SNAPSHOT - Apache NetBeans IDE 12.5
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
mavenproject16-1.0-SNAPSHOT
mavenproject17-1.0-SNAPSHOT
mavenproject18-1.0-SNAPSHOT
mavenproject19-1.0-SNAPSHOT
Web Pages
WEB-INF
beans.xml
faces-config.xml
web.xml
index.html
index.xhtml
RESTful Web Services
Source Packages
com.mycompany.mavenproject19
com.mycompany.mavenproject19.resources
kalkulator
calc.java
Test Packages
Other Sources
faces-config.xml - Navigator
version='1.0' encoding='UTF-8'
Source PageFlow History
<faces-config version="2.3"
xmlns="http://xmlns.
xmlns:xsi="http://www
xsi:schemaLocation="
<managed-bean>
<managed-bean-name>
calc
</managed-bean-name>
<managed-bean-class>
kalkulator.calc
</managed-bean-class>
<managed-bean-scope>
request
</managed-bean-scope>
</managed-bean>
</faces-config>
85

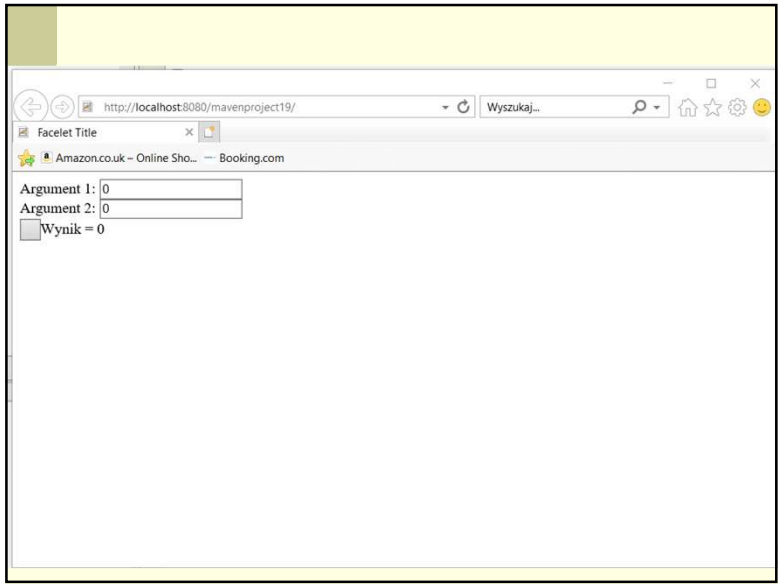

mavenproject19-1.0-SNAPSHOT
Web Pages
WEB-INF
beans.xml
faces-config.xml
web.xml
index.html
index.xhtml
RESTful Web Services
Source Packages
com.mycompany.mavenproject19
com.mycompany.mavenproject19.resources
kalkulator
calc.java
Test Packages
Other Sources
Navigator
CSS
HTML
html
XHTML
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org
<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 id="formularz">
<h:outputLabel for="formularz:wartosc1:"> Argument 1: </h:outputLabel>
<h:inputText id="wartosc1" value="#{calc.wartosc1}" />

<h:outputLabel for="formularz:wartosc2"> Argument 2: </h:outputLabel>
<h:inputText id="wartosc2" value="#{calc.wartosc2}" />

<h:commandButton label="SUMA" action="#{calc.licz}" />
<h:outputText value="Wynik = #{calc.wynik}" />
</h:form>
</h:body>
</html>
86


mavenproject19-1.0-SNAPSHOT
Web Pages
WEB-INF
beans.xml
faces-config.xml
web.xml
index.html
index.xhtml
RESTful Web Services
Source Packages
com.mycompany.mavenproject19
com.mycompany.mavenproject19.resources
kalkulator
calc.java
Test Packages
Other Sources
Dependencies
Java Dependencies
Project Files
Navigator
Members
calc
calc()
getWartosc1() : int
getWartosc2() : int
getWynik() : int
licz() : String
Source History
package kalkulator;
import javax.faces.bean.ManagedBean;
@ManagedBean
public class calc {
private int wartosc1;
private int wartosc2;
private int wynik;
public int getWartosc1(){ return wartosc1; }
public void setWartosc1(int wartosc1){
this.wartosc1=wartosc1;
}
public int getWartosc2(){ return wartosc2; }
public void setWartosc2(int wartosc2){
this.wartosc2=wartosc2;
}
Output
87


Web Pages
WEB-INF
beans.xml
faces-config.xml
web.xml
index.html
index.xhtml
RESTful Web Services
Source Packages
com.mycompany.mavenproject19
com.mycompany.mavenproject19.resources
kalkulator
calc.java
Test Packages
Other Sources
Dependencies
Java Dependencies
Project Files
Members
calc
calc()
getWartosc1() : int
getWartosc2() : int
getWynik() : int
licz() : String
setWartosc1(int wartosc1)
Source History
package kalkulator;
import javax.inject.Named;
@Named
public class calc {
private int wartosc1;
private int wartosc2;
private int wynik;
public int getWartosc1(){ return wartosc1; }
public void setWartosc1(int wartosc1){
this.wartosc1=wartosc1;
}
public int getWartosc2(){ return wartosc2; }
public void setWartosc2(int wartosc2){
this.wartosc2=wartosc2;
}
Output
88



JSF

`UI:include`

Element podobny do `jsp:include` - wykorzystywany do zamieszczenia tej samej zawartości w wielu stronach XHTML.

Atrybut `src` określa nazwę pliku (nazwa relatywna względem pliku, w którym zostanie umieszczona zawartość).

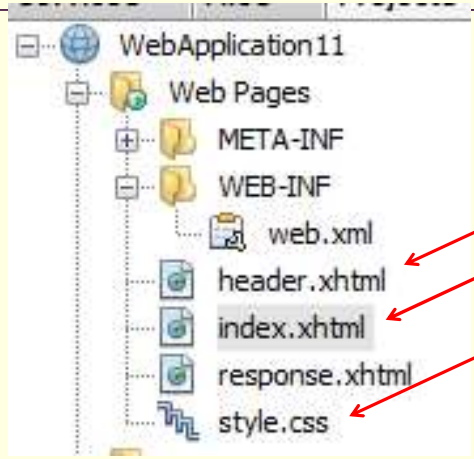
90

JSF

Przykład w NetBeans
Formatowanie CSS i dołączanie innych stron

91

JSF - kalkulator



92

style.css:

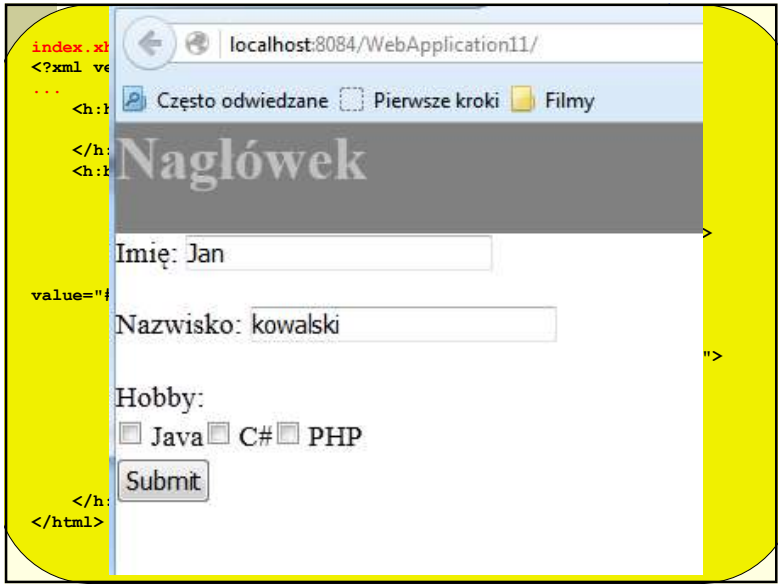
```
*{
    margin: 0;
    padding: 0;
}

#header{
    background-color: gray;
    width: 100%;
    height: 60px;
}

h1{
    color: silver;
}
```

header.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>
        <link rel="stylesheet" type="text/css"
href="style.css"/>
    </h:head>
    <h:body>
        <div id="header">
            <h1>
                Nagłówek
            </h1>
        </div>
    </h:body>
</html>
```

index.xhtml:

```
<?xml version='1.0' encoding='UTF-8' ?>
...
    <h:head>
        <title>Rejestracja studenta</title>
    </h:head>
    <h:body>
        <ui:include src="header.xhtml"/>
        <h:form>
            Imię: <h:inputText id="imie" value="#{student.imie}"/>
            <br/><br/>
            Nazwisko: <h:inputText id="nazwisko"
value="#{student.nazwisko}"/>
            <br/><br/>
            Hobby:
            <h:selectManyCheckbox value="#{student.ulubionyJezyk}">
                <f:selectItem itemValue="Java" itemLabel="Java"/>
                <f:selectItem itemValue="C#" itemLabel="C#"/>
                <f:selectItem itemValue="PHP" itemLabel="PHP"/>
            </h:selectManyCheckbox>
            <h:commandButton value="Submit" action="response"/>
        </h:form>
    </h:body>
</html>
```



JSF

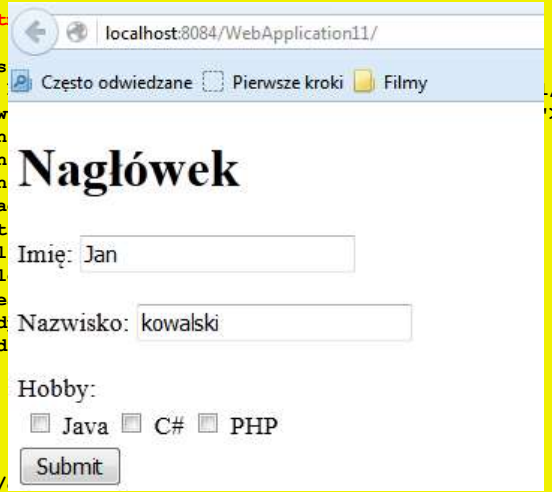
W przypadku, gdy tylko fragment kodu strony ma stanowić wstawiany element, należy ten fragment zawrzeć pomiędzy znacznikami:

```
<ui:composition>
     fragment strony
<ui:composition>
```

97

header.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:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Facelet Title</title>
        <link rel="stylesheet" type="text/css"
href="style.css"/>
    </h:head>
    <h:body>
        <div id="header">
            <ui:composition>
            <h1>
                Nagłówek
            </h1>
            </ui:composition>
        </div>
    </h:body>
</html>
```



JSF

Wykorzystanie mechanizmu szablonów

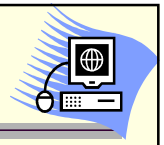
Szablony tworzą spójny widok witryny internetowej na wszystkich jej stronach

W JSF mechanizm szablonów wymaga stworzenia 2 plików:

- **Pliku szablonu,**
- **Pliku widoku.**

100

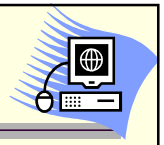
JSF



Plik szablonu:

```
<html lang=„pl”
      xmlns=„http://www.w3.org/1999/xhtml”
      xmlns:ui=„http://xmlns.jcp.org/jsf/facelets”
      xmlns:h=„http://xmlns.jcp.org/jsf/html”>

...
Wstawia zawartość o nazwie "content" umieszczoną w widoku:
<ui:insert name=„content”>Główna zawartość </ui:insert>

...
```

JSF



Plik widoku:

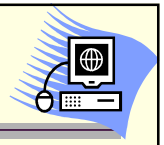
```
<html...>
  <h:body>
Definiuje szablon:
    <ui:composition template=„/template.xhtml”>
Definiuje sekcję, która zostanie wstawiona do pliku szablonu pod
nazwą "content":
      <ui:define name=„content">
        ...
      </ui:define>
    </ui:composition>
  </h:body>
</html>
```

JSF

Nawigacja pomiędzy stronami Internetowymi może zostać zakodowana w pliku `WEB-INF/faces-config.xml` jako reguły

```
<navigation-rule>
  <from-view-id>/index.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>Success</from-outcome>
    <to-view-id>/response.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/response.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>back</from-outcome>
    <to-view-id>/index.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

JSF

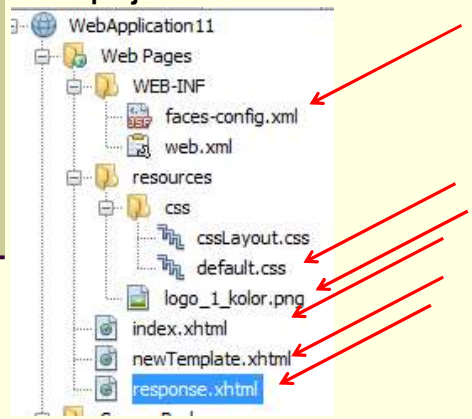


Przykład w NetBeans

104

JSF

Pliki projektu:



105

JSF

default.css:

```
#top {
    position: relative;
    background-color: greenyellow;
    color: white;
    padding: 5px;
    margin: 0px 0px 10px 0px;}
#footer {
    position: relative;
    background-color: lightgray;
    padding: 5px;
    margin: 10px 0px 0px 0px;}
#content {
    position: relative;
    background-color: gold;
    padding: 5px;
    margin: 10px 0px 0px 0px;
}
```

JSF

Plik newTemplate.xhtml

107

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
        <h:outputStylesheet name="./css/default.css"/>
        <title>Facelets Template</title>
    </h:head>
    <h:body>
        <div id="top" class="top">
            <h:graphicImage url="#{resource['logo_1_kolor.png']}"
                            width='800' height='100'
                            alt='logo PWSZ'/>
            <ui:insert name="top">Nagłówek</ui:insert>
        </div>
        <div id="content" class="center_content">
            <ui:insert name="content">Zawartość główna</ui:insert>
        </div>
        <div id="footer" class="footer">
            <ui:insert name="footer">Stopka</ui:insert>
        </div>
    </h:body>
</html>
```

index.xhtml:

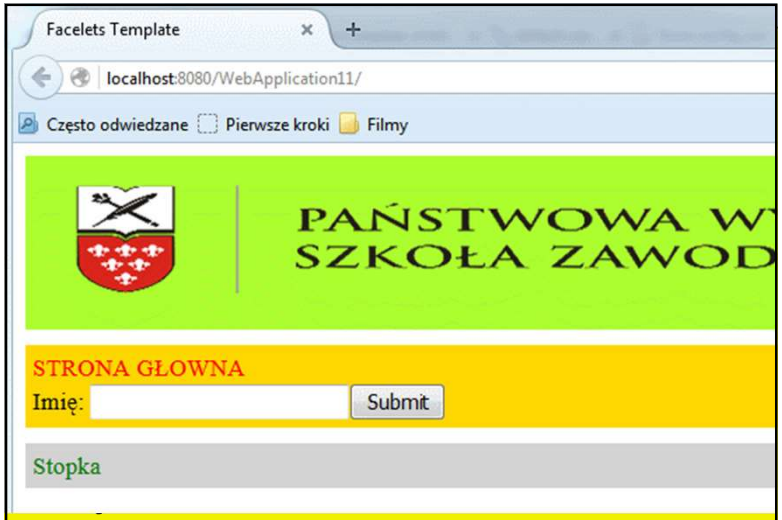
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <ui:composition template="/newTemplate.xhtml">
        <ui:define name="footer">
            <span style="color:green"> Stopka </span>
        </ui:define>
        <ui:define name="top">
            <span style=„color:blue"> Nagłówek </span>
        </ui:define>
        <ui:define name="content">
            <span style=„color:red"> STRONA GŁOWNA </span>
            <h:form>
            Imię: <h:inputText id="imie" value="#{imie}"/>
            <h:commandButton value="Submit" action="Success"/>
        </h:form>

        </ui:define>
    </ui:composition>
</html>
```

response.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:ui="http://xmlns.jcp.org/jsf/facelets">
    <ui:composition template="/newTemplate.xhtml">
        <ui:define name="footer">
         <span style="color:green"> Stopka </span>
        </ui:define>
        <ui:define name="top">
            <span style=„color:blue"> Nagłówek </span>
        </ui:define>
        <ui:define name="content">
           <span style=„color:red"> Witaj #{imie} </span>
            <h:form>
            <h:commandButton value="Cofnij" action="back"/>
            </h:form>
        </ui:define>
    </ui:composition>
</html>
```

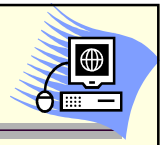
faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
              xmlns="http://xmlns.jcp.org/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">

<navigation-rule>
  <from-view-id>/index.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>Success</from-outcome>
    <to-view-id>/response.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/response.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>back</from-outcome>
    <to-view-id>/index.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>

</faces-config>
```



</faces-config>

JSF



Nawigacja dynamiczna

Niekiedy zdarza się, że nie znamy na sztywno adresu strony (nie można zapisać danych w pliku `faces-config.xml`).

Możliwe jest wykorzystaniem metody ziarna, która zwróci dynamiczny ciąg znaków w zależności od warunku.

plik User.java:

```
package nysa.pwsz;
import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

/**
 *
 * @author Administrator
 */
@ManagedBean
@SessionScoped
public class User implements Serializable {
    private String imie="Jan";
    public User(){};
    public String getImie(){return imie;}
    public void setImie(String imie){this.imie=imie;}
    public String getWynik()
    {
        if(imie.equals("Jan"))
            return "Success";
        else
            return "Fail";
    }
}
```

fragment faces-config.xml:

```
<navigation-rule>
  <from-view-id>/index.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>Success</from-outcome>
    <to-view-id>/response.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/index.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>Fail</from-outcome>
    <to-view-id>/fail.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/response.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>back</from-outcome>
    <to-view-id>/index.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

fragment index.xhtml:

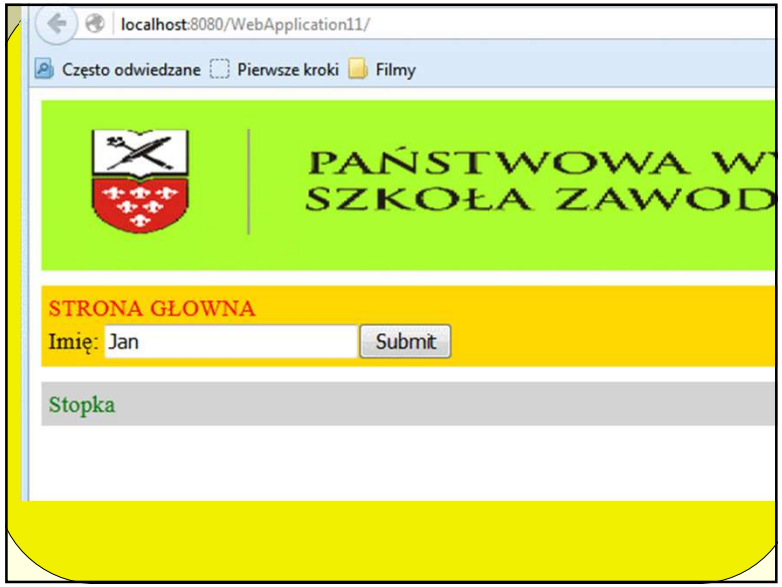
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <ui:composition template="/newTemplate.xhtml">
        <ui:define name="footer">
            <font color="green"> Stopka </font>
        </ui:define>
        <ui:define name="top">
            <font color="blue"> Nagłówek </font>
        </ui:define>
        <ui:define name="content">
            <font color="red"> STRONA GŁOWNA </font>
            <h:form>
            Imię: <h:inputText id="imie" value="#{user.imie}"/>
            <h:commandButton value="Submit"
action="#{user.getWynik()}"/>
        </h:form>

        </ui:define>
    </ui:composition>
</html>
```

fragment response.xhtml:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <ui:composition template="/newTemplate.xhtml">
        <ui:define name="footer">
            <font color="green"> Stopka </font>
        </ui:define>
        <ui:define name="top">
            <font color="blue"> Nagłówek </font>
        </ui:define>
        <ui:define name="content">
            <span style="color:red"> Witaj #{user.imie} </span>
            <h:form>
            <h:commandButton value="Cofnij" action="back"/>
            </h:form>
        </ui:define>
    </ui:composition>
</html>
```

fragment fail.xhtml:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <ui:composition template="/newTemplate.xhtml">
        <ui:define name="footer">
            <font color="green"> Stopka </font>
        </ui:define>
        <ui:define name="top">
            <font color="blue"> Nagłówek </font>
        </ui:define>
        <ui:define name="content">
  <span style="color:red"> Nie masz na imię JAN - SYNTAX ERROR :)
</span>
        </ui:define>
    </ui:composition>
</html>
```



JSF

<h:dataTable> - znacznik wykorzystywany do tworzenia tabel. Wybrane atrybuty:

Atrybut	opis
value	Kolekcja przechowująca wartości wstawiane do wierszy
var	nazwa zmiennej licznikowej (kolejne wiersze traktowane są jak kolejne przebiegi pętli)
cellpadding, cellspacing, ...	Formatowanie tabeli

<h:column>- znacznik wykorzystywany w celu załadowania danych do określonej kolumny (należy określić właściwość licznika).
Znacznik <c:facet name="header">...</c:facet> pozwala na określenie nagłówka kolumny.

120

plik Student.java:

```
public class Student {
    private int id;
    private String imie;
    private double srednia;
    public Student(int id, String imie, double srednia){
        this.id=id;
        this.imie=imie;
        this.srednia=srednia;
    }
    public int getId() {        return id;
    }
    public void setId(int id) {        this.id = id;
    }
    public String getImie() {        return imie;
    }
    public void setImie(String imie) {        this.imie = imie;
    }
    public double getSrednia() {        return srednia;
    }
    public void setSrednia(double srednia) {
        this.srednia = srednia;
    }
}
```

plik Dane.java:

```
@ManagedBean(name="tabela")
@RequestScoped
public class Dane {
    public List<Student> studentList;
    public Dane(){
        studentList= new ArrayList<>();
        studentList.add(new Student(1, "Jan", 3.0));
        studentList.add(new Student(2, "Anna", 4.0));
        studentList.add(new Student(3, "Marek", 5.0));
    }
    public List<Student> getStudentList(){
        return studentList;
    }
}
```

plik index.xhtml:

```
<h:body>
<h1><h:outputText value="Przykład wykorzystania h:dataTable:"/></h1>
<h:dataTable value="#{tabela.studentList}" var="row" border="1">
    <h:column>
        <f:facet name="header">ID</f:facet>
        <h:outputText value="#{row.id}"/>
    </h:column>
    <h:column>
        <f:facet name="header">Imie</f:facet>
        <h:outputText value="#{row.imie}"/>
    </h:column>
    <h:column>
        <f:facet name="header">Srednia</f:facet>
        <h:outputText value="#{row.srednia}"/>
    </h:column>
</h:dataTable>
</h:body>
```

JSF

cykl życia aplikacji JSF można podzielić na szereg faz:

1. Przywrócenie widoku,
2. Pobieranie danych z żądania,
3. Walidacja,
4. Aktualizacja wartości w modelu,
5. Wywołanie zadeklarowanych metod
6. Renderowanie odpowiedzi,

Klasa `javax.faces.event.PhaseListener` posiada następujące metody:
`public void beforePhase(PhaseEvent pe)` – metoda wywoływana przed rozpoczęciem każdej z faz
`public void afterPhase(PhaseEvent pe)` – po zakończeniu fazy
`public PhaseId getPhaseId()` - zwraca identyfikator fazy, w której `Listener` chce się wywołać.
`getFacesContext()` – zwraca kontekst aplikacji JSF

124

JSF

Po otrzymaniu żądania JSF rozpoczyna fazę Przywrócenia widoku.

Podczas tej fazy JSF tworzy widok strony, wiąże odpowiednie komponenty strony z klasami obsługującymi zdarzenia, walidatorami.

W przypadku, gdy żądanie do strony było pierwszym żądaniem, JSF tworzy pusty widok (zostanie on wypełniony w fazie renderowania).

Jeżeli żądanie jest typu postback, odpowiedni widok już istnieje. Podczas tej fazy jest on przywracany (wykorzystując informacje o stanie zapisane po stronie klienta lub serwera).

125

JSF

Pobieranie danych z żądania

Po przywróceniu struktury komponentów odczytywane są nowe parametry przekazane przez użytkownika.

Odczytana wartość zostaje zapisana w danym komponencie.

Jeżeli z jakiś powodów nie uda się poprawnie odczytać wartości, zostanie utworzony komunikat o błędzie po czym zostaje on wysłany do kolejki komunikatów `FacesContext`.

Zostanie on wyświetlony już w trakcie tworzenia strony HTML, łącznie z innymi komunikatami które zostały czy też zostaną utworzone w kolejnych fazach.

126

JSF

Walidacja

Następuje sprawdzenie poprawności danych zapisanych w komponentach (atrybuty są testowane pod kątem reguł oraz ograniczeń).

Wystąpienie błędu spowoduje dodanie komunikatu do kolejki komunikatów w `FacesContext`.

Jeżeli jakakolwiek metoda wywoła w tej fazie metodę `FacesContext.renderResponse`, następuje przejście do fazy renderowania.

Jeżeli aplikacja przekierowuje żądanie lub na stronie nie ma komponentów JSF, może wywołać metodę `FacesContext.responseComplete`

127

JSF

Aktualizacja wartości w modelu

Po sprawdzeniu poprawności danych następuje przypisanie wartości do pól obiektów znajdujących się na serwerze.

JSF zaktualizuje wyłącznie właściwości określone przez atrybuty komponentów wejściowych.

Jeżeli dane komponentów nie mogą zostać skonwertowane do typów właściwości ziaren następuje przejście do fazy renderowania.

Jeżeli jakikolwiek element wywoła metodę `renderResponse` nastepuje przejście do fazy renderowania.
Jeżeli aplikacja przekierowuje żądanie lub na stronie nie ma komponentów JSF, może wywołać metodę `FacesContext.responseComplete`

128

JSF

Wywołanie zadeklarowanych metod

Podczas fazy obsługiwane są wszystkie zdarzenia z poziomu aplikacji (np. zatwierdzanie formularza).

Jeżeli aplikacja wymaga przekierowania lub generuje odpowiedź nie zawierającą komponentów JSF, może wywołać metodę responseComplete.

Jeżeli przetwarzany widok jest rekonstruowany z poprzedniego żądania i jeżeli jakiś komponent wygenerował zdarzenie, zdarzenie jest przekazywane do odpowiedniego słuchacza.

129

JSF

Renderowanie odpowiedzi
Generowana jest strona JSP/XHTML.

W przypadku pierwszego żądania, komponenty znajdujące się na stronie zostają dodane do drzewa komponentów.

Jeżeli jest do żądanie postback, komponenty już się znajdują w drzewie komponentów (więc nie muszą być dodawane ponownie).

W przypadku, gdy pojawiły się jakieś błędy w poprzednich fazach, zostanie wyświetlona oryginalna strona wraz z odpowiednimi komunikatami o błędach.

Po zrenderowaniu zawartości widoku, stan odpowiedzi jest zachowywany w celu odtworzenia go w fazie przywracania widoku przy następnym żądaniu.

JSF

plik faces-config.xml:

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
              xmlns="http://xmlns.jcp.org/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
    <lifecycle>
        <phase-listener>
            jsf.MyPhaseListener
        </phase-listener>
    </lifecycle>
</faces-config>
```

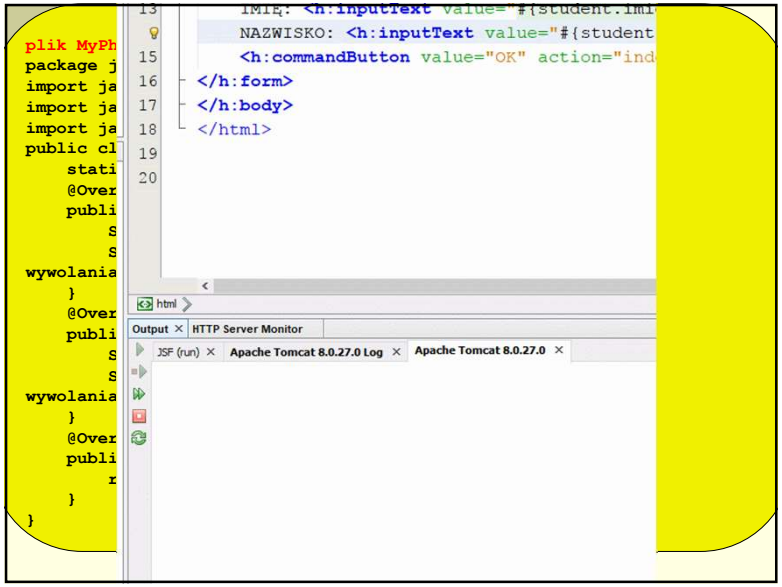
JSF

plik index.xhtml:

```
 <h:head>
        <meta charset="UTF-8"/>
        <title>Facelet Title</title>
    </h:head>
<h:body>
<h1><h:outputText value="Przykład wykorzystania PhaseListener'a"/></h1>
<h:form id='form'>
    IMIĘ: <h:inputText value="#{student.imie}"/><br/>
    NAZWISKO: <h:inputText value="#{student.srednia}"/><br/>
    <h:commandButton value="OK" action="index"/>
</h:form>
</h:body>
</html>
```

plik MyPhaseListener.java:

```
package jsf;
import javax.faces.event.PhaseEvent;
import javax.faces.event.PhaseId;
import javax.faces.event.PhaseListener;
public class MyPhaseListener implements PhaseListener{
    static int nr_wywolania=0;
    @Override
    public void afterPhase(PhaseEvent pe) {
        System.out.println("afterPhase:\n");
        System.out.println("ID: "+pe.getPhaseId()+" kolejnosc
wywolania: "+nr_wywolania+++"\n");
    }
    @Override
    public void beforePhase(PhaseEvent pe) {
        System.out.println("afterPhase:\n");
        System.out.println("ID: "+pe.getPhaseId()+" kolejnosc
wywolania: "+nr_wywolania+++"\n");
    }
    @Override
    public PhaseId getPhaseId() {
        return PhaseId.ANY_PHASE;
    }
}
```



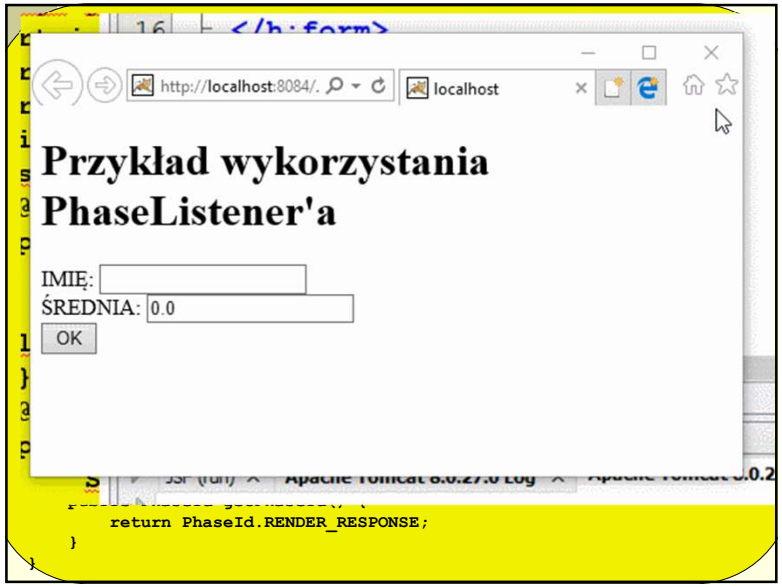
JSF

plik index.xhtml:

```
<h:body>
<h1><h:outputText value="Przykład wykorzystania PhaseListener'a"/></h1>
<h:form id='form'>
    IMIĘ: <h:inputText value="#{student.imie}"/><br/>
    ŚREDNIA: <h:inputText value="#{student.srednia}"/><br/>
    <h:commandButton value="OK" />
</h:form>
</h:body>
</html>
```

plik MyPhaseListener.java:

```
public class MyPhaseListener implements PhaseListener{
. . .
    @Override
    public void beforePhase(PhaseEvent pe) {
        UIComponent drzewo;
        drzewo= pe.getFacesContext().getViewRoot();
        przetwarzajDrzewo(drzewo);
    }
    private void przetwarzajDrzewo(UIComponent galaz){
        System.out.println(galaz.getClass().toString()+"\n");
        for(UIComponent wezel: galaz.getChildren()){
            if(wezel instanceof HtmlInputText){
                HtmlInputText it=(HtmlInputText) wezel;
                String tekst= (String) it.getValue().toString();
                if("UKRYJ".equals(tekst)){ it.setRendered(false);}
                if("ZABLOKUJ".equals(tekst)){ it.setReadonly(true); }
            }
            przetwarzajDrzewo(wezel);
        }
    }
    @Override
    public PhaseId getPhaseId() {
        return PhaseId.RENDER_RESPONSE;
    }
}
```



JSF

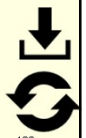
Konwersja jest wykonywana przed walidacją

- Jeżeli element XHTML posiada właściwość `immediate` ustawioną na `true`

Konwersja i walidacja są wykonywane (w czasie przypisania wartości z żądania do komponentu - 2 faza - normalnie przepisanie tylko tekstów do komponentów)

- W przeciwnym wypadku (domyślnie)

Konwersja i walidacja są wykonywane
- PO zastosowaniu wartości z żądania do komponentu oraz
- PRZED aktualizacją modelu.



138

JSF

Przetwarzanie żądania

Konwertuje żądane wartości (tekst) do wymaganych typów
wiek: `String` do `int`
cena: `String` do `double` lub `Currency`
...

Renderowanie odpowiedzi

Konwersja danych z modelu do tekstu.
wiek: `int` do `String`
cena: `double` lub `Currency` do `String` ...

139

JSF

Dwa typy konwersji:

- automatyczna
- wymuszona

Konwersja automatyczna:
konwersja tego typu zostanie automatycznie wywołana dla:
- prymitywnych typów danych i klas z nimi powiązanych (`int` i `Integer`)
- `java.math.BigDecimal`
- `java.math.BigInteger`

Konwersja wymuszona wymaga stworzenia:
- elementów `XHTML`,
- specjalnych klas konwerterów.

JSF

Konwersja automatyczna:

Wykonywana jeżeli typ wartości może zostać określony

Określenie typu odbywa się poprzez bindowanie wartości
`<h:inputText id=„age” value=„#{user.age}”/>`

typ elementu `user.age` jest określany i na podstawie tego odpowiednia konwersja automatyczna jest wykonywana

141

Ziarno:

```
@ManagedBean
@SessionScoped
public class User implements Serializable {
    private String imie="Jan";
    public String getImie() {        return imie;    }
    public void setImie(String imie) {  this.imie = imie;    }
    public String getNazwisko() {        return nazwisko;    }
    public void setNazwisko(String nazwisko) {
        this.nazwisko = nazwisko;    }
    public int getRodzenstwo() {
        return rodzenstwo;    }
    public void setRodzenstwo(int rodzenstwo) {
        this.rodzenstwo = rodzenstwo;    }
    public Date getUrodziny() {
        return urodziny;    }
    public void setUrodziny(Date urodziny) {
        this.urodziny = urodziny;    }
    private String nazwisko="Kowalski";
    private int rodzenstwo=0;
    private Date urodziny=null;
    public User(){};

}
```

index.xhtml:

```
<h:form>


<p><h:message class="error" for="imie" id="imieError" /><br/>
                Podaj imię
                <h:inputText id="imie"
                            title="Nazywasz się: "
                            value="#{user.imie}"
                            required="true"
                            requiredMessage="Wprowadź imię."
                            maxlength="30"></h:inputText>
            </p>
<p><h:message class="error" for="nazwisko" id="nazwiskoError"
/><br/>


...
```

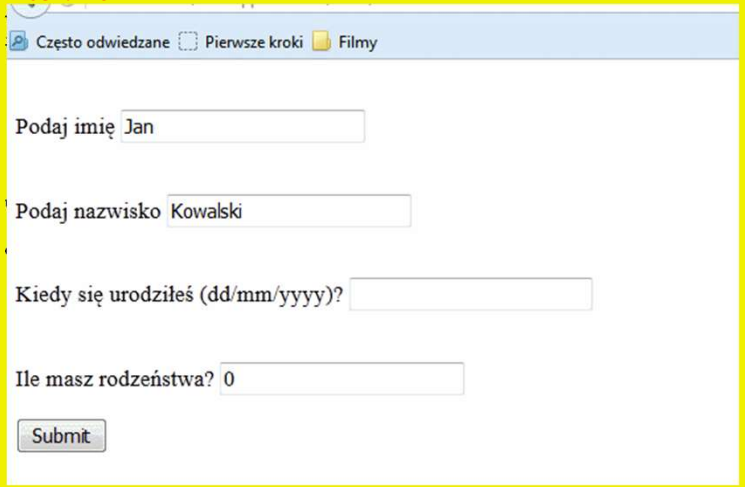
index.xhtml:

```
Podaj nazwisko
                <h:inputText id="nazwisko"
                            title="Na nazwisko masz: "
                            value="#{user.nazwisko}"
                            required="true"
                            requiredMessage="Wprowadź
nazwisko."
                            maxlength="50"></h:inputText>
            </p>
<p><h:message class="error" for="urodziny" id="urodzinyError"
/><br/>
                Kiedy się urodziłeś (dd/mm/yyyy)?
                <h:inputText id="urodziny"
                            title="Data urodzin: "
                            value="#{user.urodziny}"
                            required="true"
                            requiredMessage="Wprowadź datę
urodzin."
                            converterMessage="Wprowadź datę w
odpowiednim formacie"
                            maxlength="10"></h:inputText>
            </p>
            ...
```

index.xhtml:

```
<p><h:message class="error" for="rodzenstwo"
id="rodzenstwoError" /><br/>
                Ile masz rodzeństwa?
                <h:inputText id="rodzenstwo"
                           title="Liczba rodzenstwa: "
                           value="#{user.rodzenstwo}"
                           required="true"
                           requiredMessage="Wprowadź dane"
                           converterMessage="Wprowadź liczbę
całkowitą"
                           maxlength="2"></h:inputText>
            </p>
            <h:commandButton value="Submit" action="Success"/>

        </h:form>
```

index.xhtml:



JSF

Konwersja wymuszona

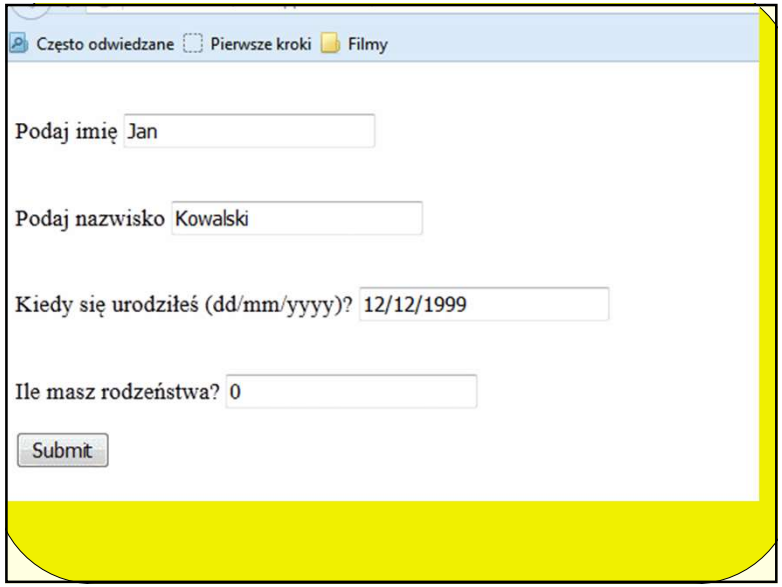
3 elementy XHTML:
1. convertDateTime
java.text.SimpleDateFormat

```
<p> Jaka jest data Twoich urodzin?
<h:inputText value=„#{user.urodziny}” id=„urodziny”>
<f:convertDateTime pattern=„dd/mm/yyyy” />
</h:inputText>
```

147

index.xhtml:

```
 <p><h:message class="error" for="urodziny" id="urodzinyError"
/><br/>
                Kiedy się urodziłeś (dd/mm/yyyy)?
                <h:inputText id="urodziny"
                           title="Data urodzin: "
                           value="#{user.urodziny}"
                           required="true"
                           requiredMessage="Wprowadź datę
urodzin."
                           converterMessage="Wprowadź datę w
odpowiednim formacie"
                           maxlength="10">
                    <f:convertDateTime pattern="dd/mm/yyyy"/>
                </h:inputText>
            </p>
```



JSF

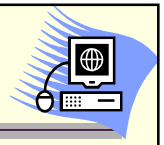
Konwersja wymuszona

3 elementy `XHTML`:
2. `convertNumber type=„percentage”/>`
typem może być również `number` lub `currency`

```
<p> Twoja roczna pęsja wynosi
<h:outputText value=„#{user.pesja}”>
<f:convertNumber type=„currency”/>
</h:outputText>
</p>
```

150

JSF



`f:convertNumber` **(klasa `NumberConverter`):**

wybrane atrybuty:

`minFractionDigits`, `maxFractionDigits` **– minimalna i maksymalna liczba cyfr po przecinku,**

`minIntegerDigits`, `maxIntegerDigits` **– minimalna i maksymalna liczba cyfr części całkowitej,**

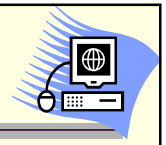
`integerOnly` (`boolean`) **– czy zwracać tylko część całkowitą,**

`type` **– dozwolone: percent (procent), currency (waluta), number (zwykła liczba),**

`pattern` **– wzorzec (zgodny z `java.text.DecimalFormat`)**

151

JSF



`Wybrane elementy z java.text.DecimalFormat:`

Znak	Opis
0	cyfra
#	cyfra - wyświetlane 0 przy braku
.	separator dziesiętny
-	znak -
,	separator grupowania
%	pomnóż przez 100 i pokaż procent
'	prefix lub suffix (wstawianie specjalnych znaków)

152

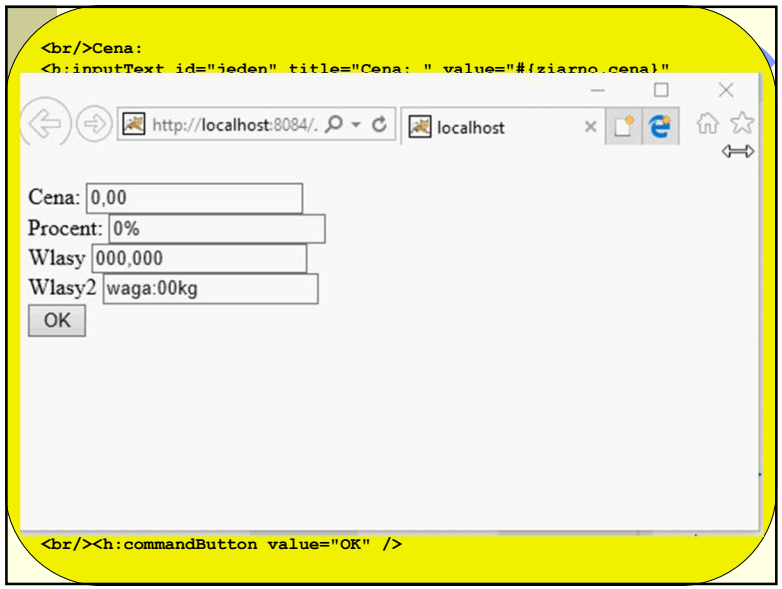
JSF

ziarno.java:

```
@ManagedBean
public class Ziarno {
    private double cena;
    private double procent;
    private double wlasny;
    private double wlasny2;
...
```

153

```
<br/>Cena:
<h:inputText id="jeden" title="Cena: " value="#{ziarno.cena}"
converterMessage="BLAD">
<f:convertNumber minFractionDigits="2"/>
</h:inputText> <h:message class="error" for="jeden" id="errJeden"
style="color:red;" />
<br/>Procent:
<h:inputText id="dwa" title="Procent " value="#{ziarno.procent}"
converterMessage="BLAD">
<f:convertNumber type="percent"/>
</h:inputText> <h:message class="error" for="dwa" id="errDwa"
style="color:red;" />
<br/>Wlasy
<h:inputText id="trzy" title="Wlasny " value="#{ziarno.wlasny}"
converterMessage="BLAD">
<f:convertNumber pattern="#000.000"/>
</h:inputText> <h:message class="error" for="trzy" id="errTrzy"
style="color:red;" />
<br/>Wlasy2
<h:inputText id="cztery" title="Wlasny " value="#{ziarno.wlasny2}"
converterMessage="BLAD">
<f:convertNumber pattern="'waga:'00'kg'"/>
</h:inputText> <h:message class="error" for="cztery" id="errCztery"
style="color:red;" />
<br/><h:commandButton value="OK" />
```



JSF

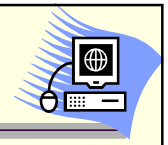
Konwersja wymuszona

3 XHTML elementy:
3. konwerter
- określenie własnego konwertera

```
<p> Podaj swoją datę urodzin
<h:inputText value=„#{user.urodziny}” id=„urodziny”>
<f:converter converterId=„nazwaKonwertera”/>
</h:inputText>
<h:message for=„urodziny”/>
</p>
```

156

JSF



Tworzenie własnego konwertera:

Klasa konwertera musi dziedziczyć po `javax.faces.convert.Converter`

Kolejno należy nadpisać metody:

`Object getAsObject(FacesContext kontekst, UIComponent komponent, String wartosc)` **– metoda zwraca obiekt przekonwertowany z podanego tekstu,**

`String getAsString(FacesContext kontekst, UIComponent komponent, Object wartosc)` **– zwraca tekst na podstawie obiektu**

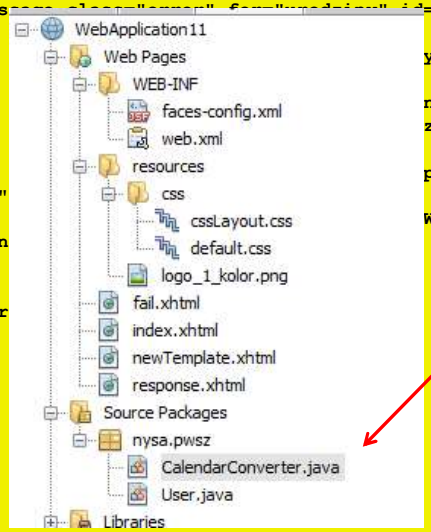
157

index.xhtml:

```
<p><h:message class="error" for="urodziny" id="urodzinyError"
/><br/>
                Kiedy się urodziłeś (dd/mm/yyyy)?
                <h:inputText id="urodziny"
                             title="Data urodzin: "
                             value="#{user.urodziny}"
                             required="true"
                             requiredMessage="Wprowadź datę
urodzin."
                             converterMessage="Wprowadź datę w
odpowiednim formacie"
                             maxlength="10">
                        <f:converter
converterId="calendarConverter"/>
                </h:inputText>
            </p>
```

index.xhtml:

```
<p><h:mes                                    ="urodzinyError"
/><br/>                                                yy)?

                                                       n: "
                                                       ziny}"

                                                       prowadź datę
urodzin."

                                                       Wprowadź datę w
odpowiedn

converter
```



CalendarConverter.java

```
package nysa.pwsz;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;
import javax.faces.convert.FacesConverter;
/**
 *
 * @author Administrator
 */
@FacesConverter(value="calendarConverter")
public class CalendarConverter implements Converter{
    private final SimpleDateFormat sdf;

...
}
```

CalendarConverter.java

```
    public CalendarConverter()
    {
        sdf=new SimpleDateFormat("dd/MM/yyyy");
        sdf.setLenient(false); //zapobiega datom typu
32.12.1999
        //dla true będzie 01.01.2000 - jeden dzień później
    }

    @Override
    public Object getAsObject(FacesContext context,
UIComponent component, String value) {
    try{
        Calendar c = Calendar.getInstance();
        c.setTime(sdf.parse(value));
        return c;
    }   catch(ParseException e)
    {
        throw new ConverterException(e);
    }
    }

...
```

CalendarConverter.java

```
    @Override
    public String getAsString(FacesContext context,
UIComponent component, Object value) {
        if(value==null)
        {
            return "";
        }

        if(value instanceof Calendar)
        {
            Calendar c = (Calendar) value;
            return sdf.format(c.getTime());
        }

        String msgDetail="Nioczekiwany typ
"+value.getClass().getCanonicalName();
        FacesMessage msg= new FacesMessage("Błąd konwersji ",
msgDetail);
        FacesContext.getCurrentInstance().addMessage(null,
msg);
        throw new ConverterException(msg);
    }
```

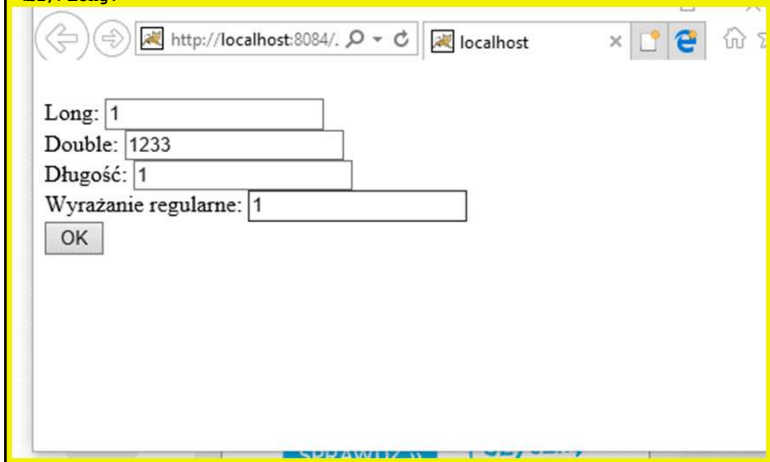
JSF

Walidatory:
JSF dostarcza zbioru standardowych klas przeznaczonych do walidacji danych:

Klasa	Tag	opis
BeanValidator	validateBean	Rejestruje walidator ziarna dla komponentu
DoubleRangeValidator	validateDoubleRange	Sprawcza czy wartość komponentu znajduje się w określonym przedziale zmiennoprzecinkowym
LengthValidator	validateLength	Sprawdza długość ciągu w komponencie
LongRangeValidator	validateLongRange	Sprawdza czy wartość komponentu znajduje się w określonym przedziale typu Long
RegexValidator	validateRegEx	Sprawdza czy wartość komponentu pasuje do wyrażenia regularnego
RequiredValidator	validateRequired	Sprawdza czy wartość komponentu nie jest pusta

```
<br/>Long:
<h:inputText id="jeden" title="wartosc1: " value="#{ziarno.wartosc1}" >
    <f:validateLongRange minimum='10' maximum='10000'/>
</h:inputText> <h:message class="error" for="jeden" id="errJeden"
style="color:red;" />
<br/>Double:
<h:inputText id="dwa" title="Procent " value="#{ziarno.wartosc2}" >
    <f:validateDoubleRange minimum='-10' maximum='10'/>
</h:inputText> <h:message class="error" for="dwa" id="errDwa"
style="color:red;" />
<br/>Długość:
<h:inputText id="trzy" title="Wlasny " value="#{ziarno.tekst1}" >
    <f:validateLength minimum='2' maximum='3'/>
</h:inputText> <h:message class="error" for="trzy" id="errTrzy"
style="color:red;" />
<br/>Wyrażanie regularne:
<h:inputText id="cztery" title="Wlasny " value="#{ziarno.tekst2}" >
    <f:validateRegex pattern="^\d{2}-\d{3}\s\b\w+\b\sP\w+a\b$"/>
</h:inputText> <h:message class="error" for="cztery" id="errCztery"
style="color:red;" />
<br/><h:commandButton value="OK" />
```

```
<br/>Long:
```



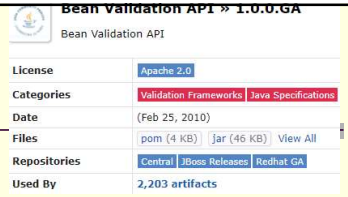
JSF

Walidacja z wykorzystaniem ziarna - system adnotacji:

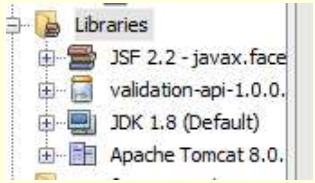
Adnotacja	Opis
`@NotNull`	nie może wystąpić wartość null
`@Size`	określa rozmiar pola (liczba znaków). Podstawowe atrybuty - min, max
`@Digits`	określa maksymalną liczbę części całkowitej i ułamkowej liczby. Podstawowe atrybuty - integer, fraction
`@DecimalMin, @DecimalMax`	określa wartość minimalną i maksymalną liczby (typ decimal)
`@Max, @Min`	ograniczenia dla liczby całkowitej
`@Pattern`	określa poprawność na podstawie zgodności z wyrażeniem regularnym
`@Past, @Future`	data musi być datą w przeszłości lub przyszłości
`@AssertTrue, @AssertFalse`	określa, że element musi przyjmować wartość true/false

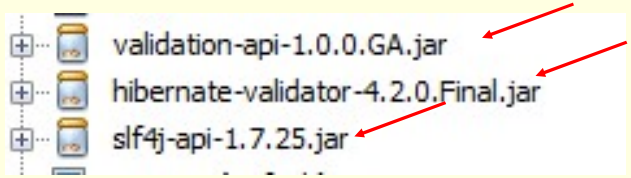
JSF

ściągnąć biblioteki:



Dodać do projektu:





167

```
@ManagedBean
@RequestScoped
public class Ziarno {
    @NotNull(message="nie może być null")
    private int id;
    @Size(min=5, max=6, message="dlugosc od 5 do 6")
    private String nazwa;
    @Digits(integer=3, fraction=3, message="minium 3 całkowite i
3 ułamkowe")
    private double liczba;
    @Pattern(regexp="^\\d\\d\\s\\w\\s[a-z]{2}$")
    private String tekst;
```

```
<h:form id="form">
<br/>ID:
<h:inputText id="jeden" title="id: " value="#{ziarno.id}" />
<h:message class="error" for="jeden" id="errJeden"
style="color:red;" />
<br/>Nazwa:
<h:inputText id="dwa" title="nazwa: " value="#{ziarno.nazwa}" />
<h:message class="error" for="dwa" id="errDwa"
style="color:red;" />
<br/>Liczba:
<h:inputText id="trzy" title="liczba: " value="#{ziarno.liczba}"
/>
<h:message class="error" for="trzy" id="errTrzy"
style="color:red;" />
<br/>Tekst:
<h:inputText id="cztery" title="tekst: " value="#{ziarno.tekst}"
/>
<h:message class="error" for="cztery" id="errCztery"
style="color:red;" />
<br/><h:commandButton value="OK" action="index.xhtml"/>
</h:form>
```