Webové aplikace 2

Java Server Faces

Martin Klíma

JAVASERVER FACES - JSF

JSF – co to je?

- Webový framework pro vytváření UI
  - server side
  - UI komponenty
  - Události
  - server-side validace
  - konverze dat
  - navigace
  - i18n
  - rozšiřitelnost
  - vlastní značky
  - propojení s server-side objekty
  - UI si pamatuje svůj stav
    - mimo jiné životní cyklus formulářů

JSF aplikace

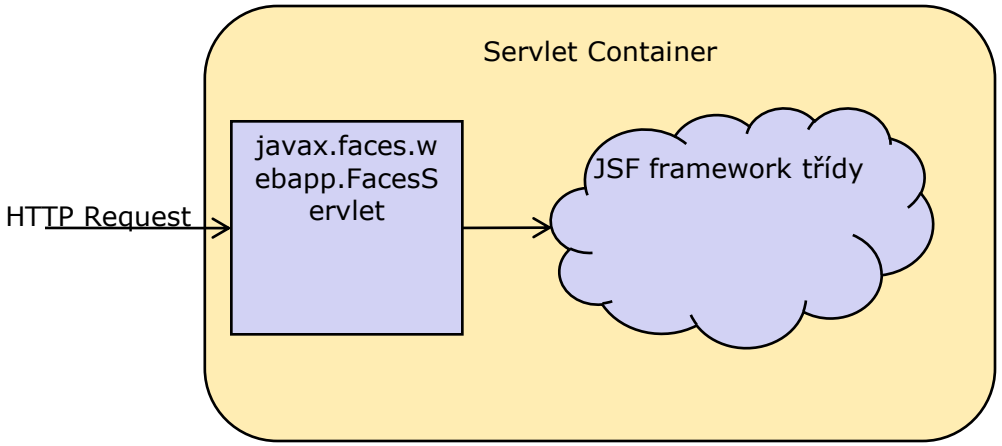
- JSP stránky
  - s vlastní sadou značek
  - sada „backing beans“
    - Java bean, které jsou spřaženy s funkčností stránky
  - konfigurační soubor
    - navigace
    - konfigurace bean, …
  - deployment descriptor
    - web.xml
  - další objekty jako validátory, konvertory, listenery, …

Postup při vývoji

1. namapování instance servletu FacesServlet
2. tvorba UI pomocí značek (i vlastních)
3. tvorba backing beans
4. definice navigace
5. přidání deklarace managed bean do konfiguračního soboru aplikace (faces-config.xml)

Servlet

- Jeden servlet funguje jako centrální controller



Web.xml

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

Tvorba UI pomocí značek

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN“  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>Člověk</title>
  </h:head>
  <h:body>
    <h1>Člověk</h1>
    <h:form>
      Křestní jméno: <h:inputText value="#{PersonManagedBean.firstName}" />
<br/>
      Příjmení: <h:inputTextarea value="#{PersonManagedBean.lastName}" />
<br/>
      Pohlaví:
      <h:selectOneRadio value="#{PersonManagedBean.gender}" title="Vyberte
pohlaví" >
        <f:selectItem itemLabel="Muž" itemValue="m"/>
        <f:selectItem itemLabel="Žena" itemValue="z"/>
      </h:selectOneRadio>
      <h:commandButton value="Odeslat"
action="#{PersonManagedBean.save}" />
    </h:form>
  </h:body>
</html>
```

Co tu vidíme

```
<h:inputText  value="#{PersonManagedBean.firstName}" />
```

Deferred volání beany PersonManagedBean

- Myšlenka: za každým formulářem je datová struktura JavaBean.
- Bean má getter, setter na všechny vlastnosti
- Bean má další metody, které můžeme volat

Použití Managed Bean

```
#{PersonManagedBean.firstName}
```

Initial response :getter

Postback response: getter

```
<h:inputText
value="#{PersonManagedBean.firstName}" />
```

Initial response: getter

Postback request: getter

Postback response: setter

Managed Bean

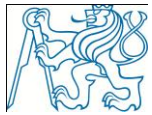
```
@ManagedBean(name="PersonManagedBean")
@SessionScoped
public class PersonManagedBean {
private String firstName;
   public PersonManagedBean() {    }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;
   }
   public String getLastName() {
      return lastName;
   }
   public String save () {
      System.out.println("First name: "+firstName);
      return "thankyou";
   }
}
```

* pro jednoduchost uvažujeme jen firstName

Jméno, kterým se na bean budeme odkazovat v JSP

@RequestScoped
@SessionScoped
@NoneScoped
@ViewScoped
@ApplicationScoped

Zápis do faces-config.xml



Managed Bean – zápis do faces-config.xml

faces-config.xml

```
<managed-bean >
      <managed-bean-name >PersonManagedBean</managed-bean-name>
      <managed-bean-class>cz.cvut.fel.beans.PersonManagedBean</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
```

* pro jednoduchost uvažujeme jen firstName

Definice navigace

- Na základě úspěchu / neúspěchu některých akcí přecházíme na další stránky
- Navigační pravidla se zapisují do faces-config.xml

```
<navigation-rule>
    <from-view-id>/person.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>thankyou</from-outcome>
        <to-view-id>/welcome.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-action>list</from-action>
        <to-view-id>/personlist.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

Z metody volané v action

```
<h:commandButton value="Odeslat"
action="#{PersonManagedBean.save}" />
```

Přímo z akce

```
<h:commandLink
value="Pryč na seznam lidí" action="list"/>
```

Dynamická navigace pomocí managedbeany

person.xhml

```
<h:commandButton value="Odeslat"
action="#{PersonManagedBean.save}" />
```

PersonManagedBean.java

```
@ManagedBean(name="PersonManagedBean")
@SessionScoped
public class PersonManagedBean {
private String firstName;
   public PersonManagedBean() {    }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;
   }
   public String getLastName() {
      return lastName;
   }
   public String save () {
      return "thankyou";
   }
}
```

faces-config.xml

```
<navigation-rule>
   <from-view-id>/person.xhtml</from-view-id>
   <navigation-case>
      <from-outcome>thankyou</from-outcome>
      <to-view-id>/welcome.xhtml</to-view-id>
   </navigation-case>
   <navigation-case>
      <from-action>list</from-action>
      <to-view-id>/personlist.xhtml</to-view-id>
   </navigation-case>
</navigation-rule>
```

Logika v Javě, můžeme vrátit různé navigační klíče

ŽIVOTNÍ CYKLUS

Konverze a validace

- Konverze
  - řeší problém převodu textového řetězce (formulář) na objekt (Java)
  - řeší i opačný převod

- Validace
  - kontrola správnosti dat

Postup

konverze -> validace

Konverze

String ⟺ Java object

Většina standardních konverzí probíhá automaticky podle datového typu managed bean.

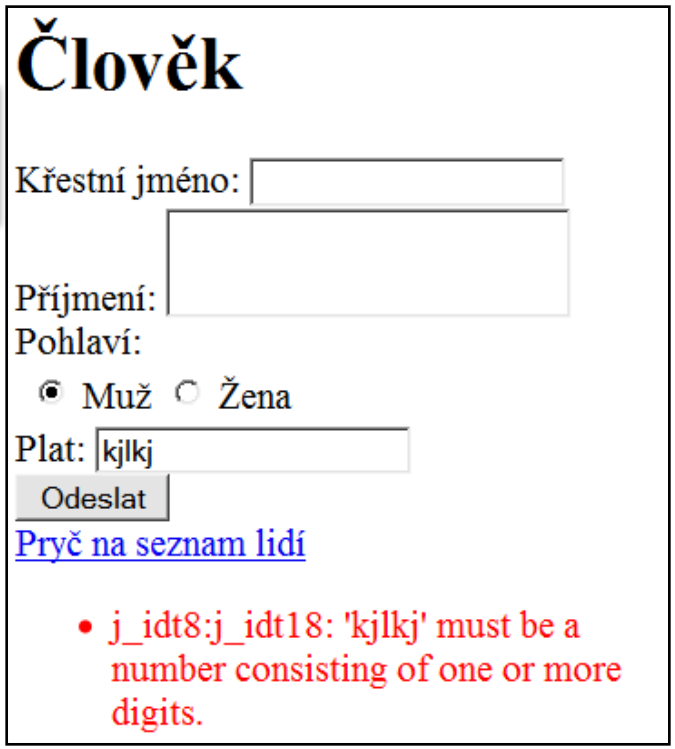
Dostupné standardní konvertory

javax.faces.BigDecimal	javax.faces.convert.BigDecimalConverter
javax.faces.BigInteger	javax.faces.convert.BigIntegerConverter
javax.faces.Boolean	javax.faces.convert.BooleanConverter
javax.faces.Byte	javax.faces.convert.ByteConverter
javax.faces.Character	javax.faces.convert.CharacterConverter
javax.faces.DateTime	javax.faces.convert.DateTimeConverter
javax.faces.Double	javax.faces.convert.DoubleConverter
javax.faces.Float	javax.faces.convert.FloatConverter

Konverze

Automatická

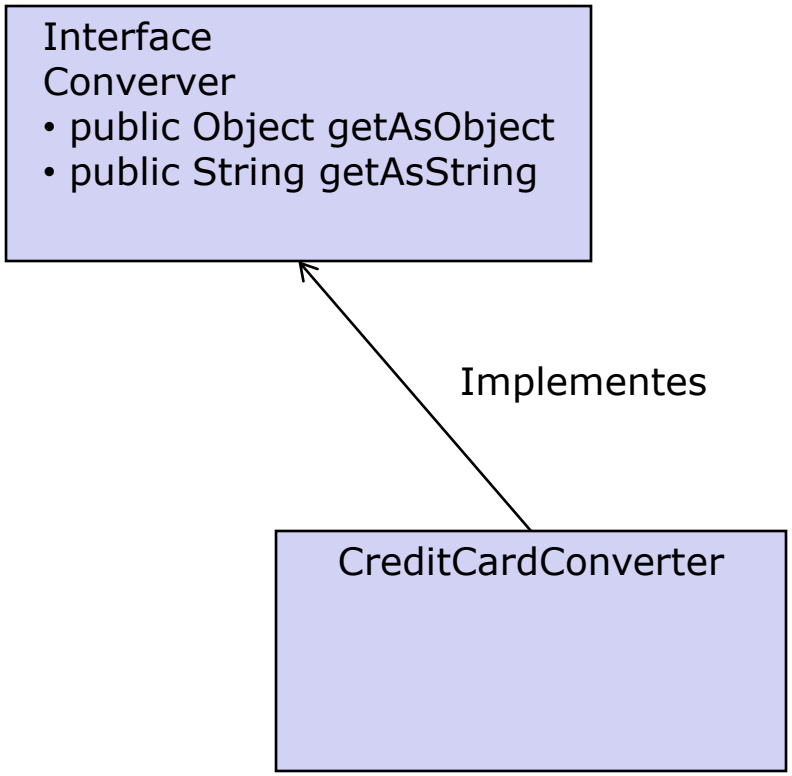
```
Plat: <h:inputText
value="#{PersonManagedBean.salary}" />
```



Explicitní

```
<h:inputText
value="#{PersonManagedBean.creditCardNr}"
                converter="CreditCardConverter" />
```

Vlastní konverter

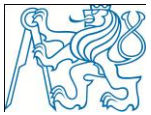


CreditCardConverter

```
public class CreditCardConverter implements Converter {
   public Object getAsObject(
       FacesContext context,
       UIComponent component,
       String newValue) throws ConverterException {

// implementace zde

       return convertedValue;
   }

   public String getAsString(
       FacesContext context,
       UIComponent component,
       Object value) throws ConverterException {

// implementace zde
       return convertedValue;
   }
}
```

faces-config.xml

```
<converter>
    <description>
       Konvertor pro preditni karty
    </description>
    <converter-id>CreditCardConverter</converter-id>
       <converter-class>
            cz.cvut.fel.converters.CreditCardConverter
        </converter-class>
  </converter>
```

Podobně jako converter funguje Validator

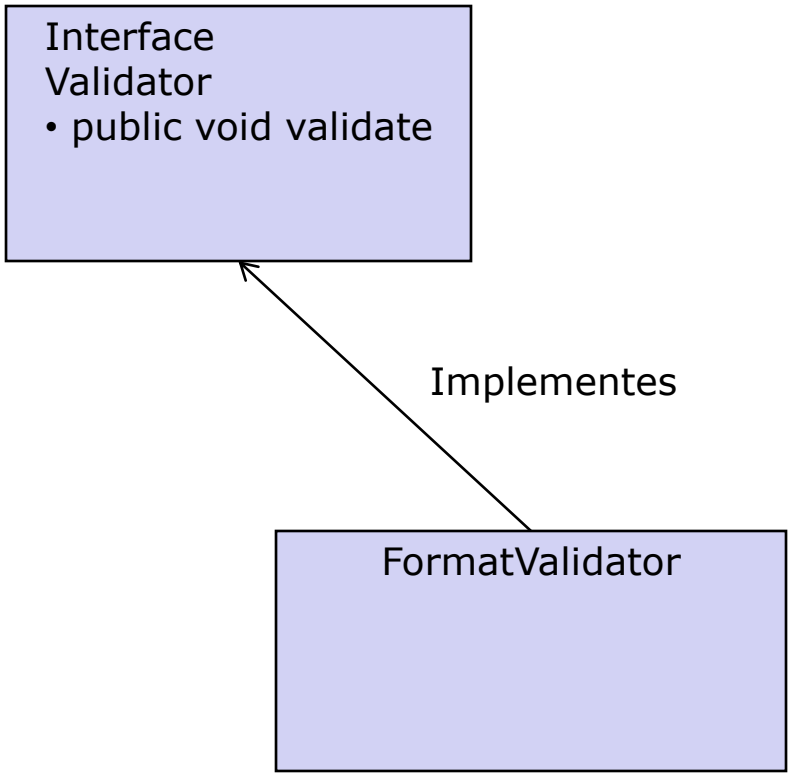
Standardní validátor

```
<h:inputText id="salary"
value="#{PersonManagedBean.salary}" >
    <f:validateLongRange maximum="100000" minimum="0" />
</h:inputText>
```

Existující validátory

- f:validateDoubleRange
- f:validateLongRange
- f:validateLength
- f:validateRegex
- f:validateRequired
- f:validateBean

Vlastní validátor



Vlastní validátor imlementace

Standardní validátor

```
@FacesValidator("formatValidator")
public class FormatValidator implements Validator {

    public void validate(FacesContext context, UIComponent
component, Object value) throws ValidatorException {
        // implementace zde
    }

}
```

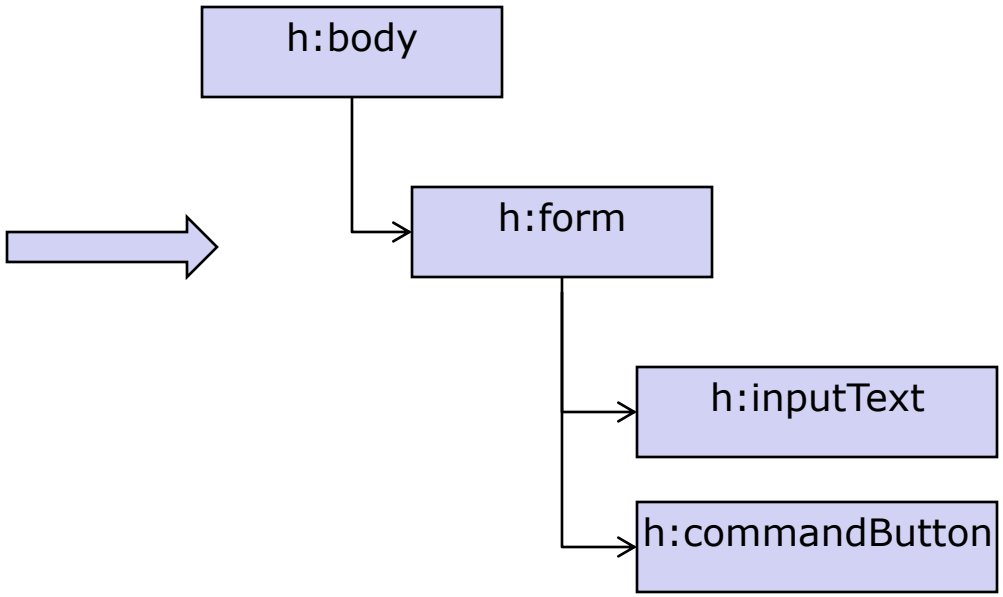
Nevalidní hodnota vyvolá vyjímu, jinak se nevrací nic

faces-config.xml

```
<validator>
    <validator-id>formatValidator</validator-id>
    <validator-class>cz.cvut.fel.validators.FormatValidator</validator-class>
</validator>
```

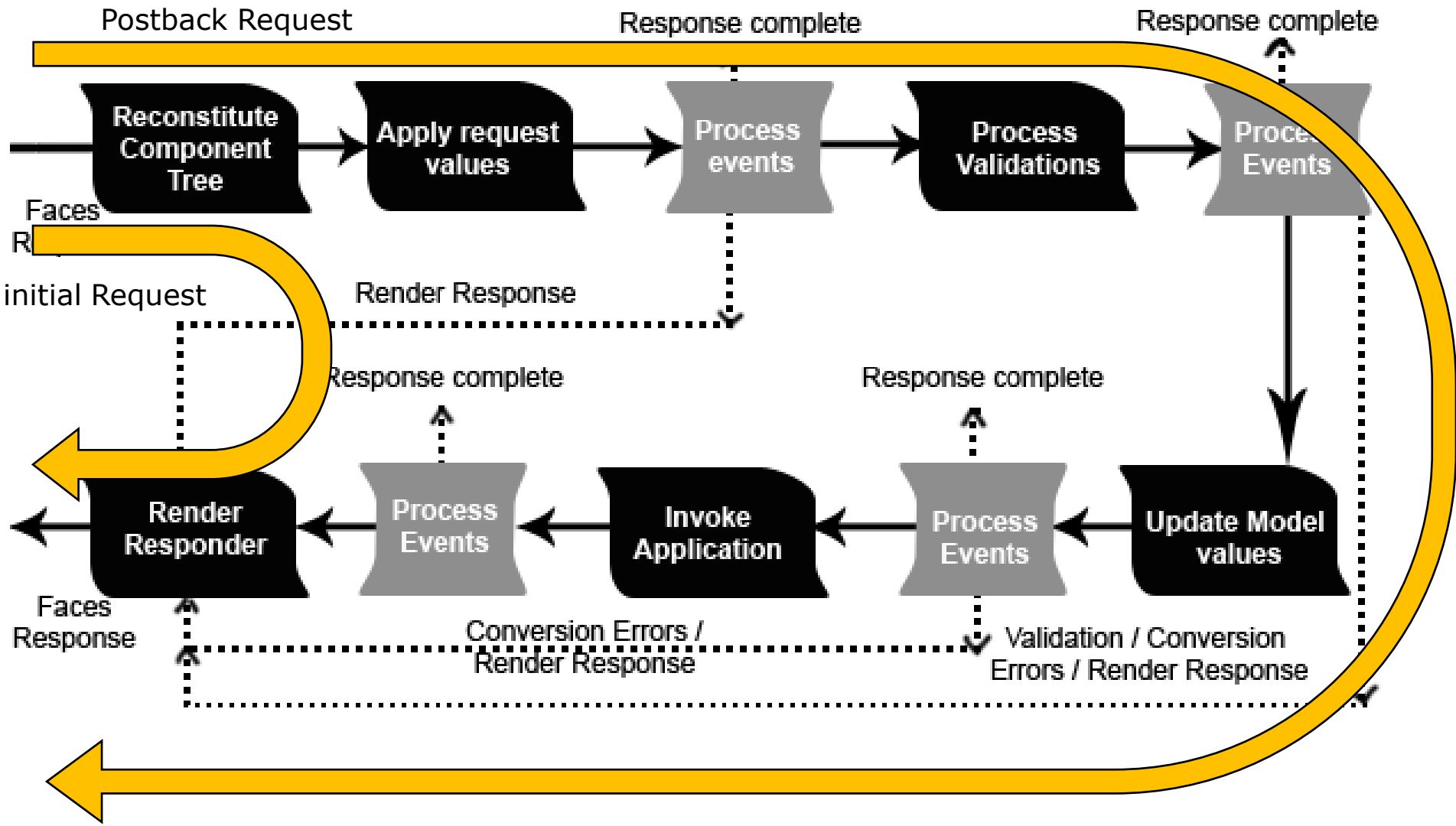
Component tree

```
<h:body>
   <h:form>
      <h:inputText />
      <h:commandButton />
   </h:form>
</h:body>
```



- Strom se staví vždy znovu
- Jednotlivé komponenty si mohou pamatovat svůj stav (stavové komponenty)
  - session
  - hidden pole, obohacování odkazů

Životní cyklus



- Navigace
  - Pokud dochází k navigaci, je přesměrování nový Initial Request!
- Konverze a Validace
  - pořadí konverze->validace
  - projdou se vždy všechny položky
  - při alespoň jedné chybě se následně skočí rovnou na Render Response
- Render response
  - vytvoření odpovědi
  - uložení stavu kompoment
    - musí implementovat rozhraní StateHolder

Interface StateHolder
- saveState
- restorState

Property immediate

- vynutíme přeskočení konverze a validace
- použití – u akcí, které nemají zpracovat data formuláře
- případně u akcí, které zpracovávají jen část dat formuláře

JSF komponenty

- Tag – značka
  - inputText,
- UIComponent
  - reprezentace ve stromu
- Renderer
  - „vykreslí“ komponentu v HTML

Reference

konverze a validace:

http://www.ibm.com/developerworks/java/library/j-jsf3/

jsf tutorial:

http://java.sun.com/javaee/javaserverfaces/reference/docs/index.html