JAVASERVER FACES
messages, validators, converters

Ненко Табаков
Пламен Танов
Технологическо училище “Електронни системи”
Технически университет – София

5 януари 2009

ЛИТЕРАТУРА
НЕОБХОДИМИ ПРОГРАМИ

- The Java EE 5 Tutorial - http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf
- JSF Tags Reference - http://java.sun.com/javaee/javaserverfaces/1.2_MR1/docs/tlddocs/inde
- Примери - http://www.exadel.com/tutorial/jsf/jsftags-guide.html
- Step-by-step tutorial - http://balusc.blogspot.com/2008/01/jsf-tutorial-with-eclipse-and-tomca
- Случаи на употреба - http://www.coreservlets.com/JSF-Tutorial/
- Допълнителни - http://www.ibm.com/developerworks/views/java/libraryview.jsp?searcl
- Java API документация - http://java.sun.com/javase/6/docs/api/
- Eclipse - www.eclipse.org
- Apache Tomcat - http://tomcat.apache.org/

ИНТЕРНАЦИОНАЛИЗАЦИЯ

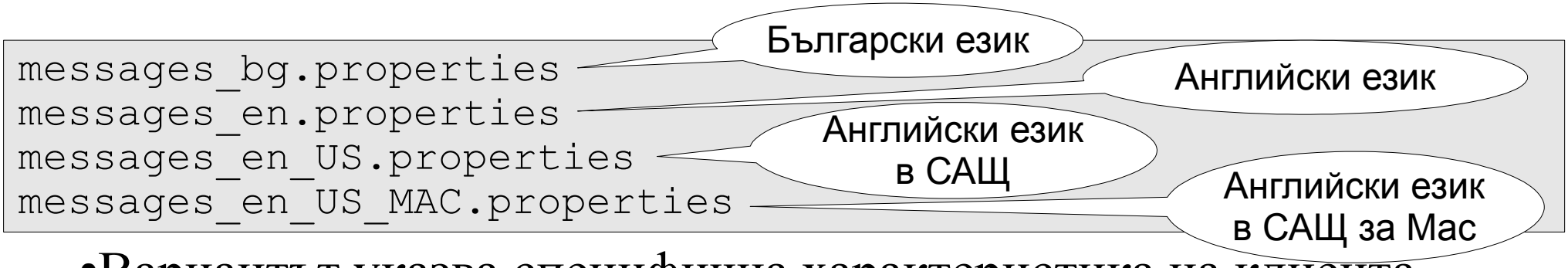
- Едно приложение би следвало да може да бъде ползвано от хора владеещи различни езици
- Един от начините за смяна на езика на приложението без да е необходимо то да се променя е посредством ползване на `ResourceBundle`
- При този подход текстът на различните съобщения в приложението се записва в отделен файл във формат:

  `ключ = стойност`
- В програмата се ползва ключът за достъп до съответния текст

```
helloButton = Say Hello!

helloWorld = Hello, {0}!
```

ИМЕНА НА ФАЙЛОВЕ

•Файловете за различните езици се кръщават съответно със суфикс, съответстващ на език, страна и вариант:



•Вариантът указва специфична характеристика на клиента

•В зависимост от езиковите настройките се избира и използва подходящият файл

•Кодовете на езиците:
http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt

•Кодовете на страните:
http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

ПАРАМЕТРИ

- В стойността може да се указват параметри, които се попълват при извличането на текста
- Това става с число, оградено с фигурни скоби
- Числото съответства на позицията на параметъра

```
helloButton = Say Hello!

helloWorld = Hello, {0}!
```

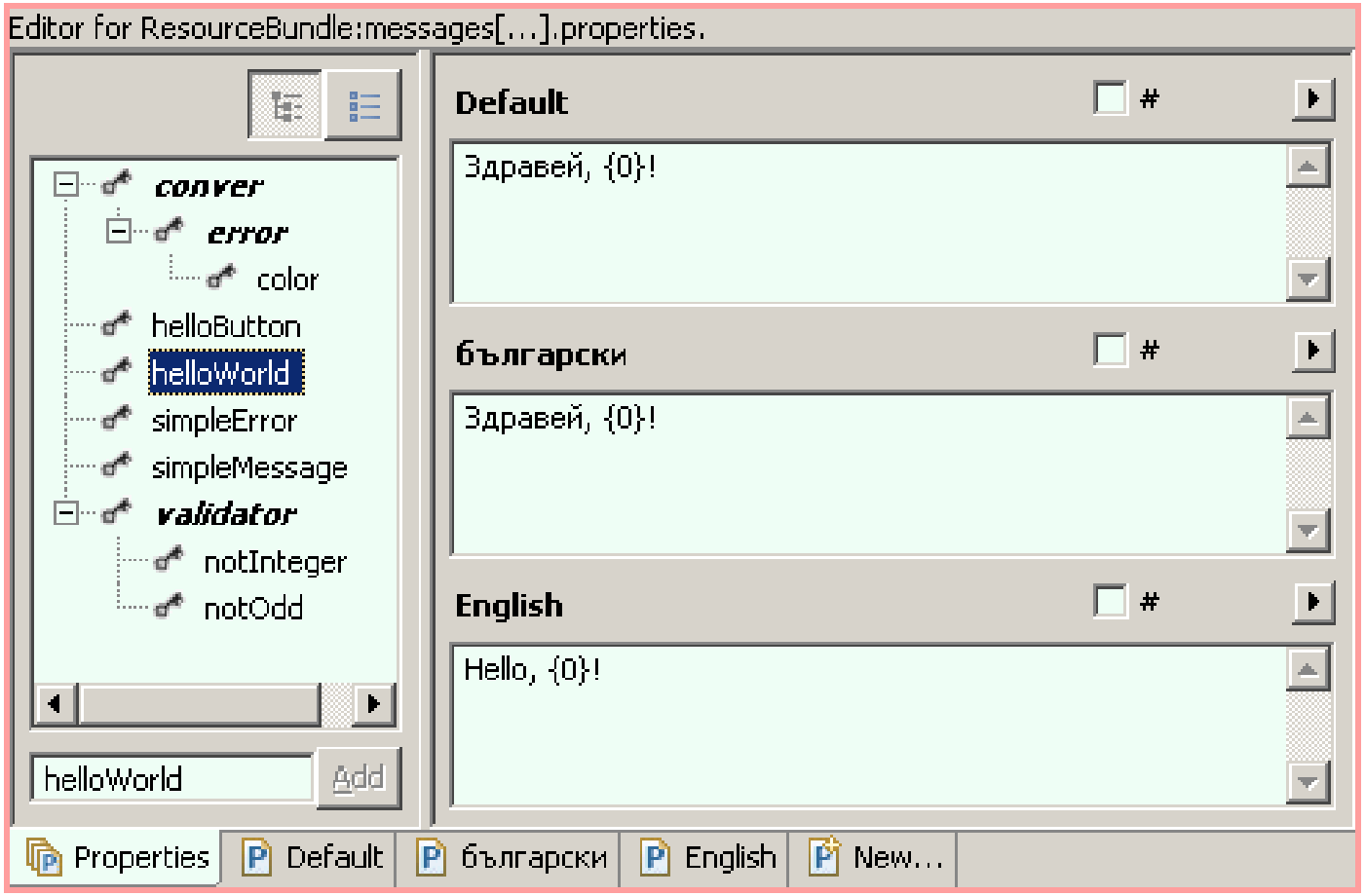
UNICODE СИМВОЛИ

- Unicode символите се кодират посредством \uXXXX
- XXXX е шестнадесетичната стойност на символа

```
#Кажи здравей!
helloButton = \u041A\u0430\u0436\u0438
                    \u0437\u0434\u0440\u0430\u0432\u0435\u0439!


#Здравей, {0}!
helloWorld = \u0417\u0434\u0440\u0430\u0432\u0435\u0439, {0}!
```

ResourceBundle Editor

- http://sourceforge.net/projects/eclipse-rbe/
- `Plugin` за `Eclipse`, който автоматично преобразува `Unicode` символите



ResourceBundle и JSF

- Дефинират се в `faces-config.xml`

```
<application>
   <message-bundle>
      org.elsysbg.courses.ip.jsf.examples.messages
   </message-bundle>
   <locale-config>
      <default-locale>bg</default-locale>
      <supported-locale>en</supported-locale>
   </locale-config>
   <resource-bundle>
      <base-name>
         org.elsysbg.courses.ip.jsf.examples.messages
      </base-name>
      <var>msg</var>
   </resource-bundle>
</application>
```

С това име се достига до съобщенията от страница

ДОСТЪП ДО ResourceBundle

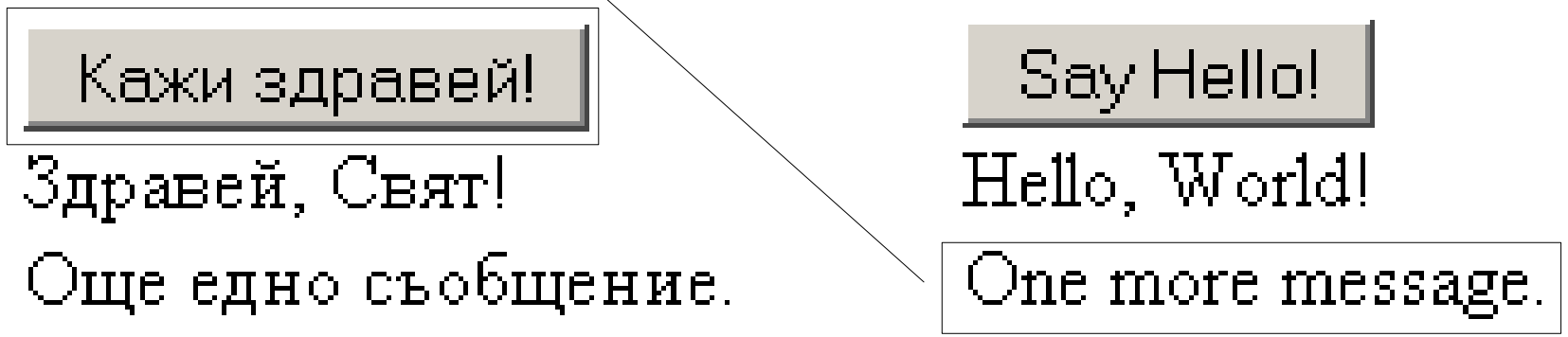
- От страница (view):

`msg` е дефинирано във `faces-config.xml`

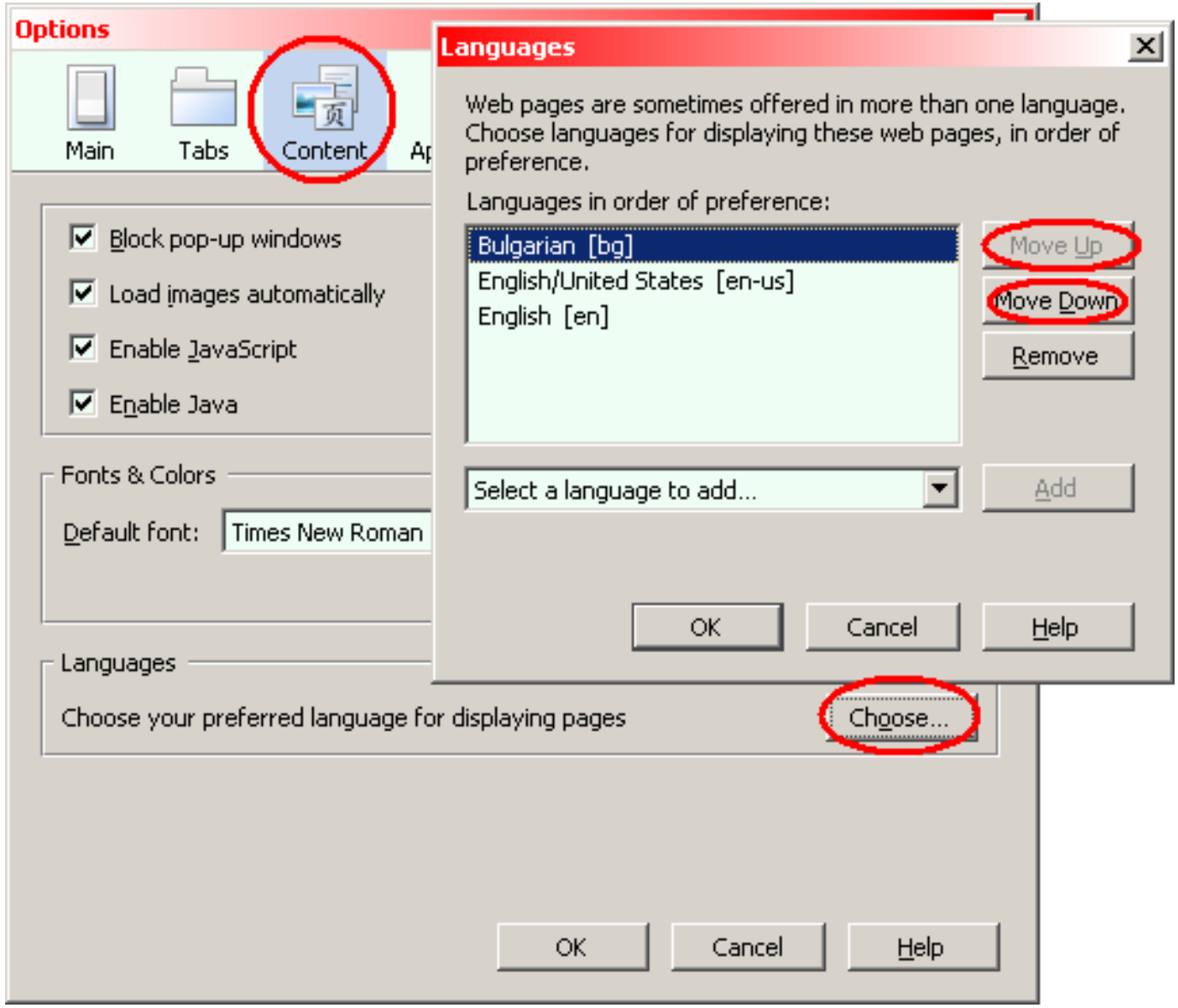
```
<h:commandButton value="#{msg.helloButton}" /> <br />

<h:outputFormat value="#{msg.helloWorld}">
   <f:param value="#{msg.world}" />
</h:outputFormat> <br />

<h:outputText value="#{msg.simpleMessage}" />
```

Здравей, {0}!
Hello, {0}!

Свят
World



ЕЗИКОВИ НАСТРОЙКИ



ДОСТЪП ДО ResourceBundle

- От код:

```
public class Messages {
   private static final String BUNDLE_NAME = "messages";

   private static final ResourceBundle RESOURCE_BUNDLE =
                        ResourceBundle.getBundle(BUNDLE_NAME);

   private Messages() {}

   public static String getString(String key) {
      try {
         return RESOURCE_BUNDLE.getString(key);
      } catch (MissingResourceException e) {
         return '!' + key + '!';
      }
   }
}
```

СЪОБЩЕНИЯ В JSF

- Потребителят трябва да бъде уведомен за изпълнението на действията предизвикани от него:
  - Дадена команда е изпълнена успешно
  - Възникнала е грешка по време на изпълнението
- За целта е дефиниран таг:

```
<h:messages
    infoClass="infoMessages"
    errorClass="errorMessages" />
```

CSS стилове при информативно съобщение и грешка

- На негово място в резултатната страница се визуализират генерираните съобщения със съответния CSS стил

ГЕНЕРИРАНЕ НА СЪОБЩЕНИЯ

- За описание на съобщението се използва клас `FacesMessage`
- Всяко съобщение има коефициент на важност (severity)

Ключ в `ResourceBundle` и параметри

```
final FacesMessage facesMessage =
                    MessageFactory.getMessage(key, attributes);

facesMessage.setSeverity(FacesMessage.SEVERITY_ERROR);


FacesContext.getCurrentInstance().addMessage(null,
                                                facesMessage);
```

Важност на съобщението

ВАЖНОСТ (SEVERITY) НА СЪОБЩЕНИЕ

Дефинирани са 4 нива:

- `FacesMessage.SEVERITY_INFO`
  - Информационно съобщение
- `FacesMessage.SEVERITY_WARN`
  - Предупредително съобщение
- `FacesMessage.SEVERITY_ERROR`
  - Съобщение за грешка
- `FacesMessage.SEVERITY_FATAL`
  - Съобщение за сериозен проблем

Посредством CSS класове може да се променя визуализацията на типовете съобщение:

- съобщенията в черно
- грешките в червено
- и другите

```
<h:messages
    infoClass="infoMessages"
    errorClass="errorMessages" />
```

ВАЖНОСТ (SEVERITY) НА СЪОБЩЕНИЕ

Съобщението беше добавено успешно.

Posts

ID	User	Text	Date
1	admin	new post	Sun Jan 04 17:57:01 EET 2009

Add post

text:	
	Add Post from admin

ВАЖНОСТ (SEVERITY) НА СЪОБЩЕНИЕ

Такъв пост вече съществува.

Posts

ID	User	Text	Date
1	admin	new post	Sun Jan 04 17:57:01 EET 2009

Add post

text:	new post Такъв пост вече съществува.
	Add Post from admin

СЪОБЩЕНИЕ ЗА КОМПОНЕНТ

- Ако съобщението е логически свързано с даден компонент на страницата:
  - Грешка при валидация на данните на компонента
- За целта е дефиниран таг:

```
<h:form id="MessageForm">
   <h:inputText value="#{bean.name}" id="name"/>
   <h:message for="name"
      infoClass="infoMessages" errorClass="errorMessages" />
</h:form>
```

- На негово място в резултатната страница се визуализира генерираното съобщение със съответния CSS стил

СЪОБЩЕНИЕ ЗА КОМПОНЕНТ

Такъв пост вече съществува.

Posts

ID	User	Text	Date
1	admin	new post	Sun Jan 04 17:57:01 EET 2009

Add post

text:	new post	Такъв пост вече съществува.
	Add Post from admin	

СЪОБЩЕНИЕ ЗА КОМПОНЕНТ

- При добавяне на съобщение се указва за кой компонент е
- Името на компонента се записва във формат:
  - име-на-форма:име-на-компонент
- Ако съобщението е глобално (не е обвързано с компонент) се записва `null`

```
FacesContext.getCurrentInstance().addMessage("MessageForm:name",
                                                facesMessage);

FacesContext.getCurrentInstance().addMessage(null,facesMessage);
```

ГЛОБАЛНИ СЪОБЩЕНИЯ

- За да се показват само глобалните съобщения (тези, които не са обвързани с компонент) се използва атрибутът `globalOnly`
- Стойността му по подразбиране е `false`, т.е. показват се всички съобщения

```
<h1>Only global messages:</h1>
<h:messages infoClass="infoMessages"
                errorClass="errorMessages" globalOnly="true"/>

<h1>All messages:</h1>
<h:messages infoClass="infoMessages"
                                    errorClass="errorMessages" />
```

ТЕКСТ НА СЪОБЩЕНИЯ

- Всяко съобщение има детайлен текст и обобщение
- По подразбиране:
  - messages показва краткия текст
  - message показва детайлния текст

```
final FacesMessage message = new FacesMessage();
message.setDetail("details message");
message.setSummary("summary message");

FacesContext.getCurrentInstance().addMessage("MessageForm:name",
                                                    message);
```

```
<h:messages infoClass="infoMessages"
                                        errorClass="errorMessages" />
<h:form id="MessageForm">
   <h:inputText id="name"/>
   <h:message for="name" infoClass="infoMessages"
                                errorClass="errorMessages" /><br />
</h:form>
```

ТЕКСТ НА СЪОБЩЕНИЯ

- Всяко съобщение има детайлен текст и обобщение
- По подразбиране:
  - messages показва краткия текст
  - message показва детайлния текст

ТЕКСТ НА СЪОБЩЕНИЯ

- Всяко съобщение има детайлен текст и обобщение
- Атрибути служат за промяна на поведението по подразбиране:
  - `showDetail`
  - `showSummary`

```
final FacesMessage message = new FacesMessage();
message.setDetail("details message");
message.setSummary("summary message");

FacesContext.getCurrentInstance().addMessage("MessageForm:name",
                                                    message);
```

```
<h:messages showDetail="true" showSummary="false"
            infoClass="infoMessages" errorClass="errorMessages" />
<h:form id="MessageForm">
   <h:inputText id="name"/>
   <h:message for="name" showDetail="false" showSummary="true"
           infoClass="infoMessages" errorClass="errorMessages"  />
</h:form>
```

ТЕКСТ НА СЪОБЩЕНИЯ

- Всяко съобщение има детайлен текст и обобщение
- Атрибути служат за промяна на поведението по подразбиране:
- `showDetail`
- `showSummary`

СПИСЪК СЪС СЪОБЩЕНИЯ

•По подразбиране списъка със съобщения се изобразява като HTML списък (`<ul> <li>`)

- Здравей, Свят!
- Още едно съобщение.

СПИСЪК СЪС СЪОБЩЕНИЯ

- По подразбиране списъка със съобщения се изобразява като HTML списък (`<ul> <li>`)
- Това може да се промени посредством:
  - CSS

Здравей, Свят!

Още едно съобщение.

```
.errorMessages {
    color: red;
    list-style-type: none;
}
.infoMessages {
    color: black;
    list-style-type: none;
}
```

СПИСЪК СЪС СЪОБЩЕНИЯ

- По подразбиране списъка със съобщения се изобразява като HTML списък (`<ul> <li>`)
- Това може да се промени посредством:
  - CSS
  - Атрибут `layout`

Здравей, Свят!

Още едно съобщение.

```
<h:messages layout="table"
        infoClass="infoMessages" errorClass="errorMessages" />
```

ВАЛИДИРАНЕ НА ВХОДНИ ДАННИ

- Преди да бъдат обработени данните е необходимо да се провери дали те са правилни
- Към всеки компонент за въвеждане на данни може да се прикачи валидатор
- Ако данните не са валидни страницата се презарежда със съответното съобщение за грешка

```
<h:inputText id="text" value="#{validateStandardRequest.text}"
label="name" required="true">
   <f:validateLength minimum="2" maximum="4"/>
</h:inputText>
<h:message for="text" errorClass="errorMessages" /> <br />

<h:inputText id="number"
      value="#{validateStandardRequest.number}" label="years">
   <f:validateLongRange maximum="100" minimum="18" />
</h:inputText>

<h:message for="number" errorClass="errorMessages" /> <br />
```

тип String

валидатор за дължина на низ

целичислен тип

валидатор за число в интервал

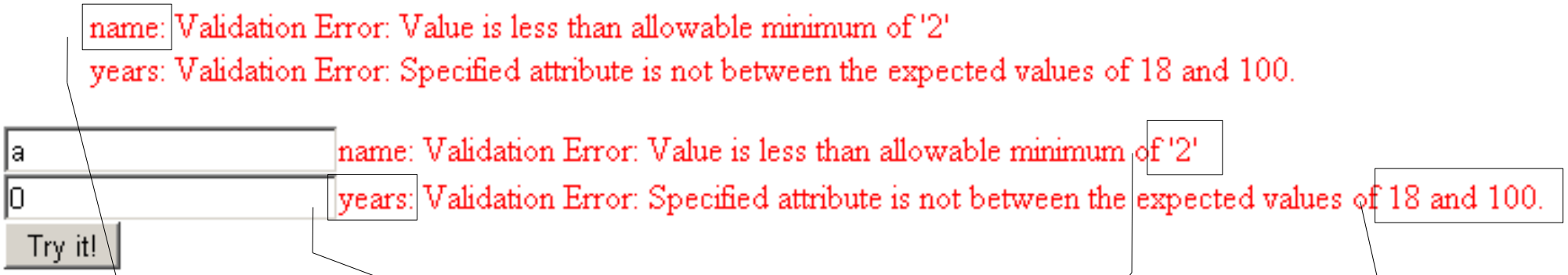
ВАЛИДИРАНЕ НА ВХОДНИ ДАННИ

- Преди да бъдат обработени данните е необходимо да се провери дали те са правилни
- Към всеки компонент за въвеждане на данни може да се прикачи валидатор
- Ако данните не са валидни страницата се презарежда със съответното съобщение за грешка

name: Validation Error: Value is less than allowable minimum of '2'
years: Validation Error: Specified attribute is not between the expected values of 18 and 100.

a name: Validation Error: Value is less than allowable minimum of '2'
0 years: Validation Error: Specified attribute is not between the expected values of 18 and 100.
Try it!

ВАЛИДИРАНЕ НА ВХОДНИ ДАННИ




```
<h:inputText id="text" value="#{validateStandardRequest.text}"
label="name" required="true">
   <f:validateLength minimum="2" maximum="4"/>
</h:inputText>
<h:message for="text" errorClass="errorMessages" /> <br />

<h:inputText id="number"
      value="#{validateStandardRequest.number}" label="years">
   <f:validateLongRange maximum="100" minimum="18" />
</h:inputText>

<h:message for="number" errorClass="errorMessages" /> <br />
```

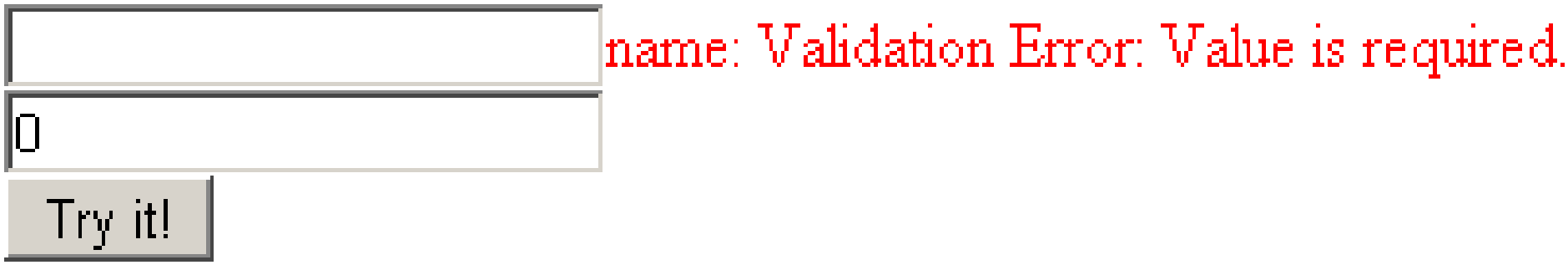
ВАЛИДИРАНЕ НА ВХОДНИ ДАННИ

- Атрибутът `label` се ползва при създаването на съобщението за грешка
- Ако той липсва на негово място се ползва името на компонента във формат:
  - име-на-форма:име-на-компонент
- Може да се използва и `ResourceBundle`:

```
<h:inputText id="text" value="#{addPostRequest.text}"
                    required="true" label="#{msg.text_label}" >
   <f:validator validatorId="DuplicatePostValidator" />
</h:inputText>
```

ЗАДЪЛЖИТЕЛНО ПОЛЕ

- Атрибутът `required` указва, че потребителят трябва да въведе стойност в съответното поле
- Ако стойност не е въведена страницата се презарежда със съответното съобщение за грешка




```
<h:inputText id="text" value="#{validateStandardRequest.text}"
                                label="name" required="true" />

<h:message for="text" errorClass="errorMessages" />
```

СЪЗДАВАНЕ НА ВАЛИДАТОР

- **JSF** предоставя механизъм за създаване на нови валидатори от потребителя
- За целта трябва да се реализира интерфейса `Validator`
- Ако дадената стойност не е валидна трябва да се генерира изключение от тип `ValidatorException`

```
public class OddValidator implements Validator {
   public void validate(FacesContext context, UIComponent
         toValidate, Object value) throws ValidatorException {
      if(!(value instanceof Integer)) {
         final FacesMessage message = /* ... */
         throw new ValidatorException(message);
      }
      final Integer number = (Integer) value;
      if(number%2 == 0) {
         final FacesMessage message = /* ... */
         throw new ValidatorException(message);
      }
   }
}
```

СЪЗДАВАНЕ НА ВАЛИДАТОР

- **JSF** предоставя механизъм за създаване на нови валидатори от потребителя
- За целта трябва да се реализира интерфейса `Validator`
- Ако дадената стойност не е валидна трябва да се генерира изключение от тип `ValidatorException`
- Валидаторът трябва да се регистрира във `faces-config.xml`

```
<validator>
   <validator-id>OddValidator</validator-id>
   <validator-class>
    org.elsysbg.courses.ip.jsf.examples.validator.OddValidator
   </validator-class>
</validator>
```

СЪЗДАВАНЕ НА ВАЛИДАТОР

- **JSF** предоставя механизъм за създаване на нови валидатори от потребителя
- За целта трябва да се реализира интерфейса `Validator`
- Ако дадената стойност не е валидна трябва да се генерира изключение от тип `ValidatorException`
- Валидаторът трябва да се регистрира във `faces-config.xml`
- Да се укаже, че съответното поле ще бъде валидирано от специфичния валидатор

```
<h:inputText id="number1"
                          value="#{validateOddRequest.number1}">
   <f:validator validatorId="OddValidator"/>
</h:inputText>
```

ВАЛИДИРАНЕ НА ВХОДНИ ДАННИ

- Поле може да се валидира и от метод
- Важат същите правила като при реализирането на интерфейса `Validator`

```
public class ValidateOddRequest {
    public void validateOdd(FacesContext context, UIComponent
                                        toValidate, Object value) {
    ...
    }
}
```

```
<h:inputText id="number2"
    value="#{validateOddRequest.number2}"
    validator="#{validateOddRequest.validateOdd}"
/>
```

ПРЕОБРАЗУВАНЕ (КОНВЕРТИРАНЕ) НА ДАННИ

- **JSF** предоставя набор от конвертори за стандартните типове данни, в това число и дата
- С тяхна помощ елементите на HTML форма се преобразуват към желания тип
- Ако това не е възможно страницата се презарежда със съответното съобщение за грешка

years: 'текст' must be a number consisting of one or more digits.

text

текст years: 'текст' must be a number between -2147483648 and 2147483647 Example: 9346

Try it!

```
<h:inputText value="#{validateStandardRequest.number}" />
```

Целочислен тип

СЪЗДАВАНЕ НА КОНВЕРТОРИ

- Потребителят може да дефинира нови типове данни (класове)
- **JSF** предоставя механизъм за създаване на конвертори от потребителя
- С тяхна помощ новите типове данни могат се визуализират със стандартните HTML тагове
- За целта трябва да се реализира интерфейсът `Converter`
- Ако дадената стойност не е валидна трябва да се генерира изключение от тип `ConverterException`
- Трябва да се създадат методи за преобразуване на обекта към низ и обратно
- Ако обектите се съхраняват в БД е възможно този низ да е първичният ключ на записа и след това обектът да се извлича от БД наново

СЪЗДАВАНЕ НА КОНВЕРТОРИ

```
public class ColorConverter implements Converter {

   public Object getAsObject(FacesContext context, UIComponent
                                          component, String value) {
      if(value==null) {
         return null;
      }

      final Color result;
      if(value.equals("red")) {
         result = new Color(255, 0, 0);
      } else if(value.equals("green")) {
         result = new Color(0, 255, 0);
      } else {
         final FacesMessage facesMessage = /* ... */
         throw new ConverterException(facesMessage);
      }
      return result;
   }
```

СЪЗДАВАНЕ НА КОНВЕРТОРИ

```
    public String getAsString(FacesContext context, UIComponent
                                        component, Object value) {
        if (value instanceof Color) {
            final Color color = (Color) value;

            return color.getName();
        }

        final String message = "This is not color: "+value;
        throw new ConverterException(message);
    }

}
```

СЪЗДАВАНЕ НА КОНВЕРТОРИ

```
    public String getAsString(FacesContext context, UIComponent
                                        component, Object value) {
        if (value instanceof Color) {
            final Color color = (Color) value;

            return color.getName();
        }

        final String message = "This is not color: "+value;
        throw new ConverterException(message);
    }

}
```

ИЗПОЛЗВАНЕ НА КОНВЕРТОРИ

- Съществуват няколко начина за използване на конвертори създадени от потребителя
- Дефиниране на конвертор за всички обекти от даден тип
  - Във `faces-config.xml`:

```
<converter>
   <display-name>ColorConverter</display-name>
   <converter-for-class>
                org.elsysbg.courses.ip.jsf.examples.model.Color
   </converter-for-class>
   <converter-class>
  org.elsysbg.courses.ip.jsf.examples.converter.ColorConverter
   </converter-class>
</converter>
```

Конвертор за:

Реализацията на конвертор

  - В страницата:

```
<h:inputText id="color" value="#{convertColorRequest.color}" />
```

ИЗПОЛЗВАНЕ НА КОНВЕРТОРИ

- Съществуват няколко начина за използване на конвертори създадени от потребителя
- Дефиниране на `ID` за конвертор
  - Във `faces-config.xml`:

```
<converter>
   <display-name>ColorConverter</display-name>
   <converter-id>ColorConverter</converter-id>
   <converter-class>
   org.elsysbg.courses.ip.jsf.examples.converter.ColorConverter
   </converter-class>
</converter>
```

ID на конвертор

  - В страницата:

```
<h:inputText id="color2" value="#{convertColorRequest.color2}"
                                converter="ColorConverter" />
```

Указване на конвертор посредством ID

ИЗПОЛЗВАНЕ НА КОНВЕРТОРИ

- Съществуват няколко начина за използване на конвертори създадени от потребителя
- Дефиниране на `ID` за конвертор
  - Във `faces-config.xml`:

```
<converter>
   <display-name>ColorConverter</display-name>
   <converter-id>ColorConverter</converter-id>
   <converter-class>
   org.elsysbg.courses.ip.jsf.examples.converter.ColorConverter
   </converter-class>
</converter>
```

ID на конвертор

  - В страницата:

```
<h:inputText id="color" value="#{convertColor...quest.color}" >
   <f:converter converterId="ColorConverter" />
</h:inputText>
```

Указване на конвертор посредством ID

ПРИМЕР

```
<h:form>
<!-- using default converter for type color, specified in
faces-config.xml -->
   <h:inputText id="color1"
      value="#{convertColorRequest.color1}" />
   <h:outputText style="color: #{convertColorRequest.color1};"
                                          value="COLOR" /><br />

   <h:inputText id="color2"
      value="#{convertColorRequest.color2}"
      converter="ColorConverter" />
   <h:outputText style="color: #{convertColorRequest.color2};"
                                          value="COLOR" /><br />

   <h:inputText id="color3"
      value="#{convertColorRequest.color3}" >
      <f:converter converterId="ColorConverter" />
   </h:inputText>
   <h:outputText style="color: #{convertColorRequest.color3};"
                                          value="COLOR" /><br />
</h:form>
```

ПРИМЕР

СТАНДАРТНИ СЪОБЩЕНИЯ ЗА ГРЕШКА

- Стандартни ключове при грешка
  - javax.faces.component.UIInput.CONVERSION
  - javax.faces.component.UIInput.REQUIRED
  - javax.faces.component.UISelectOne.INVALID
  - javax.faces.component.UISelectMany.INVALID
  - javax.faces.validator.NOT_IN_RANGE
  - javax.faces.validator.DoubleRangeValidator.MAXIMUM
  - javax.faces.validator.DoubleRangeValidator.MINIMUM
  - javax.faces.validator.DoubleRangeValidator.TYPE
  - javax.faces.validator.LengthValidator.MAXIMUM
  - javax.faces.validator.LengthValidator.MINIMUM
  - javax.faces.validator.LongRangeValidator.MAXIMUM
  - Javax.faces.validator.LongRangeValidator.MINIMUM
  - javax.faces.validator.LongRangeValidator.TYPE

СЪОБЩЕНИЯ, ВАЛИДАТОРИ и КОНВЕРТОРИ

Други примери могат да бъдат намерени на:

- http://www.jsf-faq.com/faqs/faces-messages.html
- http://www.ibm.com/developerworks/java/library/j-jsf3/