Методы блока проверки допустимости

Мы можем иначе реализовать пользовательскую проверку допустимости, добавляя методы проверки допустимости в один или более именованных бинов приложения. Следующий класс Java иллюстрирует реализацию таких методов:

\src\java\beans\AlphaValidator.java

```
package beans;

import javax.enterprise.context.RequestScoped;
import javax.faces.application.FacesMessage;
import javax.inject.Named;
import javax.faces.component.UIComponent;
import javax.faces.component.html.HtmlInputText;
import javax.faces.context.FacesContext;
import javax.faces.validator.ValidatorException;
import org.apache.commons.lang.StringUtils;

/**
 *
 * @author Computer School
 */
@Named
@RequestScoped
public class AlphaValidator  {

   public void validateAlpha(FacesContext context, UIComponent component, Object value) throws
ValidatorException {
      if (!StringUtils.isAlphaSpace((String) value)) {
         HtmlInputText htmlInputText = (HtmlInputText) component;
         FacesMessage facesMessage = new FacesMessage(htmlInputText.getLabel() + ": допустимы только
буквенные символы.");
         throw new ValidatorException(facesMessage);
      }
   }

}
```

В этом примере класс содержит только метод блока проверки допустимости. Мы можем дать нашему методу блока проверки допустимости любое имя на свое усмотрение. Однако возвращаемое им значение должно быть пустым; кроме того, три входных параметра, показанные в примере, должны использоваться в указанном порядке. Другими словами, за исключением имени метода сигнатура метода блока проверки допустимости должна быть идентичной сигнатуре метода validate(), определенного в интерфейсе javax.faces.validator.Validator.

Тело этого метода блока проверки допустимости почти идентично телу нашего метода validate() в нестандартном элементе верификации. Мы проверяем значение, вводимое пользователем, чтобы удостовериться, что оно содержит только буквенные символы и/или пробелы. Если это не так, мы вызываем исключение ValidatorException, передавая экземпляр FacesMessage, содержащий соответствующую строку сообщения об ошибке.

StringUtils

В примере мы использовали валидатор org.apache.commons.lang.StringUtils для выполнения фактической логики проверки допустимости. В дополнение к методу, используемому в примере, данный класс содержит несколько методов, позволяющих выяснить, какой является строка – числовой или алфавитно-цифровой. Этот класс, являющийся частью библиотеки Apache commons-lang, очень полезен при написании нестандартных элементов верификации.

Поскольку каждый метод проверки должен находиться в именованном бине, класс с этим методом следует декорировать аннотацией @Named, как поясняется в нашем примере.

Последнее, что нам необходимо сделать для использования нашего метода блока проверки допустимости, – связать его с нашим компонентом с помощью атрибута validator:

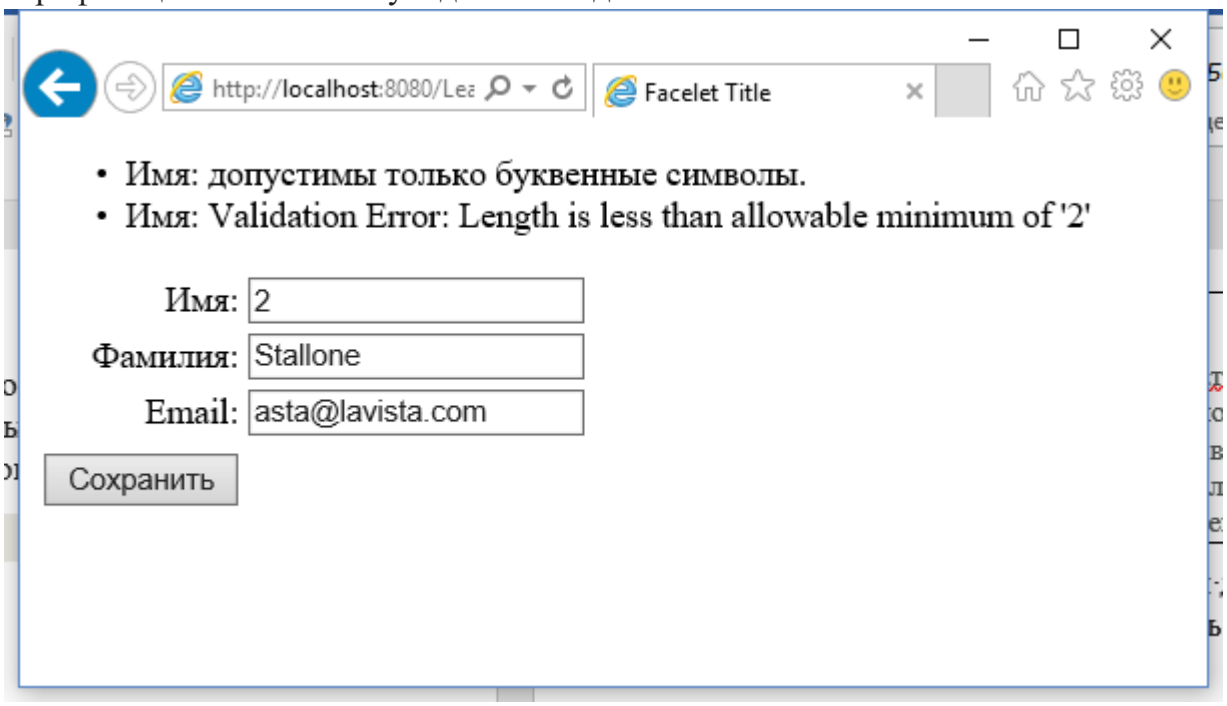
\web\index.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
   <h:head>
      <title>Facelet Title</title>
      <h:outputStylesheet library="css" name="styles.css"/>
   </h:head>
   <h:body>

      <h:form>
         <h:messages></h:messages>
         <h:panelGrid columns="2" columnClasses="rightAlign,leftAlign">
            <h:outputText value="Имя:"></h:outputText>
            <h:inputText label="Имя" value="#{customer.firstName}"
                     required="true" validator="#{alphaValidator.validateAlpha}">
               <f:validateLength minimum="2" maximum="30">
               </f:validateLength>
            </h:inputText>
            <h:outputText value="Фамилия:"></h:outputText>
            <h:inputText label="Фамилия" value="#{customer.lastName}"
                     required="true" validator="#{alphaValidator.validateAlpha}">
               <f:validateLength minimum="2" maximum="30">
               </f:validateLength>
            </h:inputText>
            <h:outputText value="Email:"></h:outputText>
            <h:inputText label="Email" value="#{customer.email}">
               <f:validator validatorId="emailValidator"/>
            </h:inputText>
            <h:panelGroup></h:panelGroup>
            <h:commandButton action="confirmation" value="Сохранить">
            </h:commandButton>
         </h:panelGrid>
      </h:form>
   </h:body>
</html>
```

Ни поле **Имя** (First Name), ни поле **Фамилия** (Last Name) не должны принимать что-либо кроме буквенных символов или пробелов, поскольку мы добавили наш метод проверки.

Обратите внимание, что значение атрибута validator блока проверки допустимости тега <h:inputText> является языком выражений JSF, по умолчанию использующим имя именованного бина, содержащего наш метод проверки допустимости. Наш бин имеет имя alphaValidator, а метод нашего блока проверки допустимости называется validateAlpha.
После изменения страницы для использования нашего нестандартного элемента верификации мы сможем увидеть его в действии:



Обратите внимание, что для поля **Имя** (First Name) оказались выполненными и наше сообщение нестандартного элемента верификации, и стандартный блок проверки допустимости длины.

Реализация методов блока проверки допустимости имеет преимущество, связанное с отсутствием издержек, возникающих при создании целого класса только для единственного метода блока проверки допустимости .При использовании классов блока проверки допустимости несколько тегов <f:validator> могут быть вложены в тег, который будет проверен, поэтому для поля может быть выполнено несколько проверок допустимости – как пользовательских, так и стандартных.