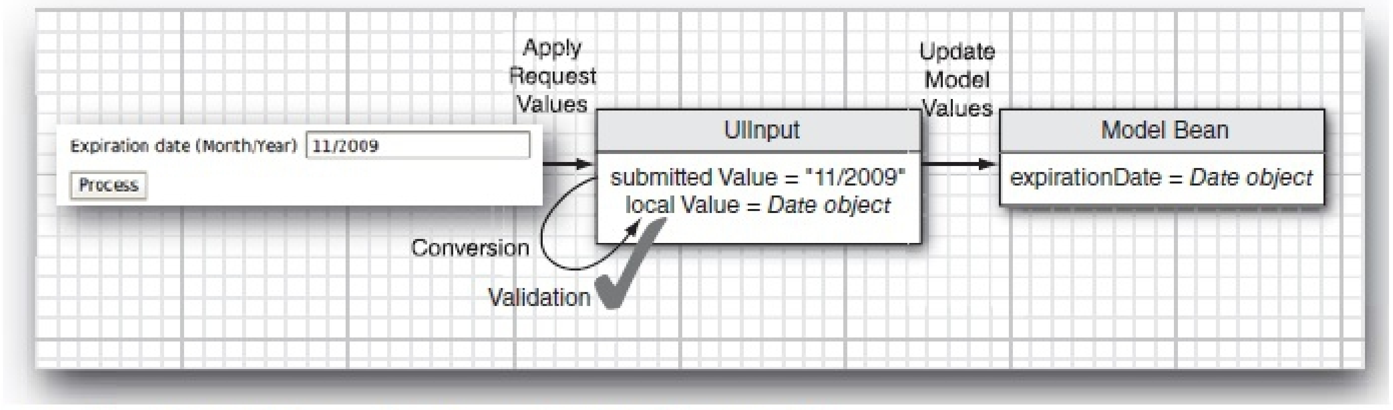
Chapter 7 Conversion and Validation

1. Introduction

➔ Lets look at how user input(form data) travels from browser to bean

1. Browser sends user input to the server using HTTP request. These are called request values.
2. Each input tag in JSF implementation has corresponding component object. In 'Apply Request Value Phase' these request values gets stored in component objects. Here they are called submitted values.
3. All request values from browser are Strings and bean properties could be int, String or any other type. Hence these values needs to be converted to a particular type and validated. Both conversion and validation processes starts with 'Process Validation' phase.
4. Once the values are converted they are not directly transferred to the bean. Before that they needed to be validated. Hence they are stored in component objects as local values. Application designers provides validation conditions.
5. After all validations are successful 'Update Model Values' phase will starts and local values are stored in beans, as specified by their value references.

Example



Expiration date (Month/Year) 11/2009
Process
Apply Request Values
UIInput
submitted Value = "11/2009"
local Value = Date object
Conversion
Validation
Update Model Values
Model Bean
expirationDate = Date object

A value travels from the browser to the model

2. Using Standard Converters

converter	Adds an arbitrary converter to a component.
convertDateTime	Adds a datetime converter to a component.
convertNumber	Adds a number converter to a component.

➔ Web application stores data in many types but web user interface deals exclusively with Strings. For example, Suppose user needs to edit a data object stored in business logic. For that first the date object will be converted to String and then sent to web browser to be displayed inside the text field. After user edited the text field, the resulting String returns to the server and must be converted to the Date object.

➔ There are three ways one can use converters

i. Using particular conversion tags

```
<f:convertNumber>
<f:convertDateTime>
```

➔ Supporse we have a shopping cart application where customer needs to fill amount, credit card number and valid date for the card. Then the converters are used in customerDetails.xhtml page are shown below

```
<h:inputText value="#{payment.amount}">
<f:convertNumber minFractionDigits="2"/>
</h:inputText>
```

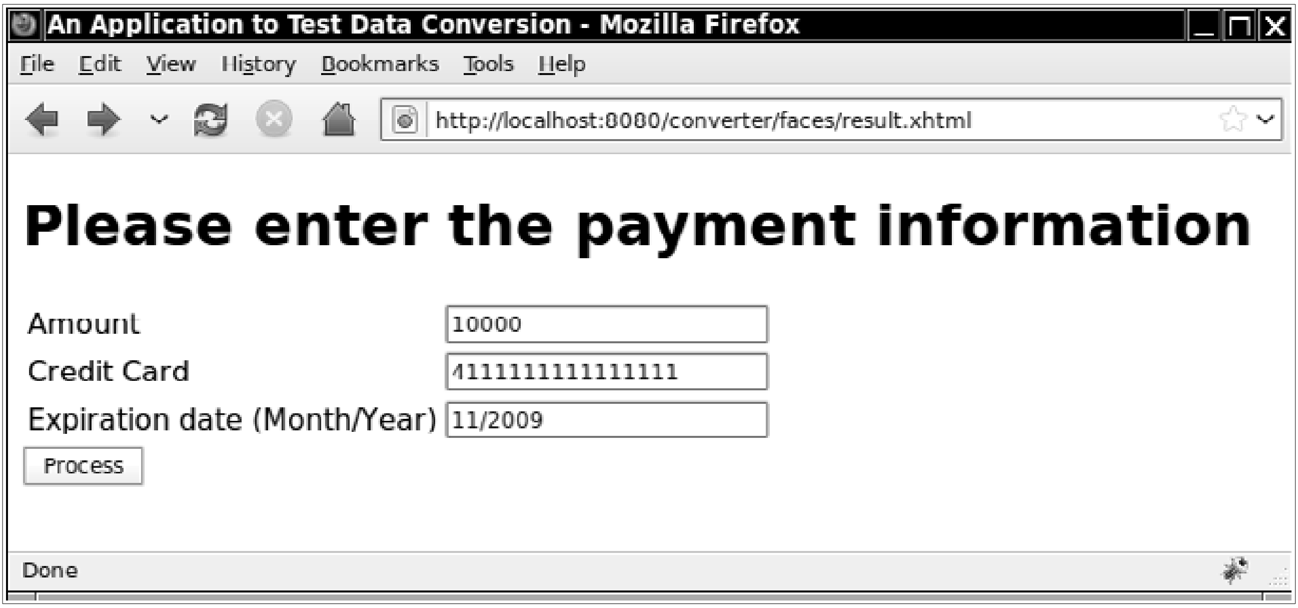
and

```
<h:inputText value="#{payment.date}">
<f:convertDateTime pattern="MM/yyyy"/>
</h:inputText>
```

In the result.xhtml page we can again use converter for 'amount' as shown below

```
<h:outputText value="#{payment.amount}">
<f:convertNumber type="currency"/>
</h:outputText>
```

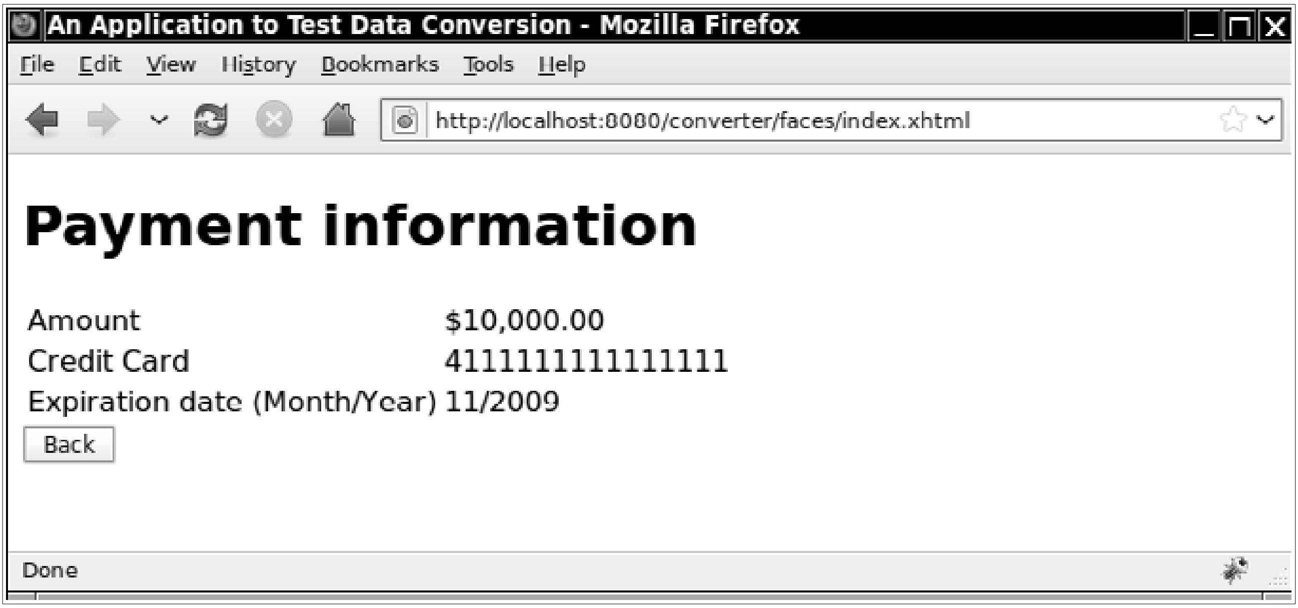
The output are as shown below


An Application to Test Data Conversion - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Process
Done

http://localhost:8080/converter/faces/result.xhtml

Please enter the payment information

Amount 10000
Credit Card 4111111111111111
Expiration date (Month/Year) 11/2009


An Application to Test Data Conversion - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Back
Done

http://localhost:8080/converter/faces/index.xhtml

Payment information

Amount $10,000.00
Credit Card 4111111111111111
Expiration date (Month/Year) 11/2009

➔ If value expression of primitive type, enumerated type or BigInteger/BigDecimal type then one dont need to provide converter as JSF implementation automatically picks a standard converter.

➔ <f:convertNumber> and <f:convertDateTime> attributes

Attributes of the f:convertNumber Tag

Attribute	Type	Value
type	String	number (default), currency, or percent
pattern	String	Formatting pattern, as defined in java.text.DecimalFormat
maxFractionDigits	int	Maximum number of digits in the fractional part
minFractionDigits	int	Minimum number of digits in the fractional part
maxIntegerDigits	int	Maximum number of digits in the integer part
minIntegerDigits	int	Minimum number of digits in the integer part
integerOnly	boolean	True if only the integer part is parsed (default: false)
groupingUsed	boolean	True if grouping separators are used (default: true)
locale	java.util.Locale or String	Locale whose preferences are to be used for parsing and formatting
currencyCode	String	ISO 4217 currency code, such as USD or EUR, for selecting a currency converter
currencySymbol	String	This string is passed to DecimalFormat.setDecimalFormatSymbols, overriding the locale-based symbol; not recommended—use currencyCode instead

Attributes of the f:convertDateTime Tag

Attribute	Type	Value
type	String	date (default), time, or both
dateStyle	String	default, short, medium, long, or full
timeStyle	String	default, short, medium, long, or full
pattern	String	Formatting pattern, as defined in java.text.SimpleDateFormat
locale	java.util.Locale or String	Locale whose preferences are to be used for parsing and formatting
timeZone	java.util.TimeZone	Time zone to use for parsing and formatting; if you do not supply a time zone, the default is GMT Note: As of JSF 2.0, you can change the default to TimeZone.getDefault() by setting javax.faces.DATETIMECONVERTER_DEFAULT_TIMEZONE_IS_SYSTEM_TIMEZONE to true in web.xml.

ii. Using 'converter' attribute

➔ One can use 'converter' attribute available with the tags.

```
<h:outputText value="#{payment.date}" converter="javax.faces.DateTime"/>
```

which is equivalent to

```
<h:outputText value="#{payment.date}">
<f:convertDateTime/>
</h:outputText>
```

iii. Use <f:converter> tag

```
<h:outputText value="#{payment.date}">
<f:converter converterId="javax.faces.DateTime"/>
</h:outputText>
```

➔ All JSF implementations must define a set of converters with predefined IDs as shown below

• `javax.faces.DateTime` (used by `f:convertDateTime`)

• `javax.faces.Number` (used by `f:convertNumber`)

• `javax.faces.Boolean`, `javax.faces.Byte`, `javax.faces.Character`, `javax.faces.Double`, `javax.faces.Float`, `javax.faces.Integer`, `javax.faces.Long`, `javax.faces.Short` (automatically used for primitive types and their wrapper classes)

• `javax.faces.BigDecimal`, `javax.faces.BigInteger` (automatically used for BigDecimal/BigInteger)

2.1 Conversion Errors

➔ When the conversion error occurs, JSF implementation carries out the following actions.

i. Component whose conversion fails, posts a message and declares itself invalid.

ii. The JSF implementation re-displays the current page immediately after 'Process Validations' phase has completed. The redisplayed page contains all the values that user provided – no user input is lost.

➔ Whenever user provides an illegal input like for integer value String has been entered etc. The JSF implementtaion automatically re-displays the current page giving the user chance to re-enter the correct values.

2.3. Displaying Error Messages

➔ By default JSF implementation displays error messages whenever conversion error occurs.

➔ For example if we have code

```
<h:outputLabel value="Amount :"/>
   <h:inputText id="amt" value="#{customer.amount}">
      <f:convertNumber minFractionDigits="2"/>
   </h:inputText>
```

'amout' is of type int in customer bean and if 'ten' has been entered then the JSF implementation re-displays current page with an error message at the bottom 'j_idt5:amt: 'ten' is not a number.'

➔ One can customize error messages of a page using `<h:message>` and `<h:messages>` tags.

➔ If one want to display error message next to the component that reported them then give an ID to that component and reference that ID in `<h:message>` as shown below.

```
<h:outputLabel value="Amount :"/>
                    <h:inputText id="amt" label="Amount is invalid"
```

```
                                        value="#{customer.amount}">
                                        <f:convertNumber minFractionDigits="2"/>
                                    </h:inputText>
                                    <h:message for="amt"/>
```

Error message will be 'Amount is Invalid: 'ten' is not a number. Example: 99'

➔ JSF implementation shows default(standard) error message because child tag <h:messages/> will be automatically added to the view if the project stage has been set to development(web.xml).

➔ Error messages actually has two parts 'summary' and 'detail'. By default, <h:message> displays 'detail' and hides the summary and <h:messages> displays summary and hides the detail.

➔ In the above example <h:message> is showing 'detail' error message. 'detail' error message shows the label of the component, the offending value, and a sample of a correct value.(note: 'label' has been added in <h:inputText> tag).

➔ 'summary' message omits the sample.

➔ <h:message> and <h:messages> have attributes 'showDetails' and 'showSummary'. Both these attributes are of type Boolean and one can display summary or detail for an error message by setting their values to true or false. Example

```
<h:outputLabel value="Amount :"/>

<h:inputText id="amt" label="Amount is Invalid" value="#{customer.amount}">
   <f:convertNumber minFractionDigits="2"/>
</h:inputText>
<h:message for="amt" showSummary="true" showDetail="false"/>
```

The error message will be 'Amount is Invalid: 'ten' is not a number.'

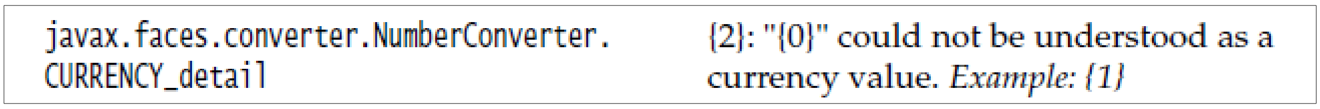
➔ One can even use the color of an error message using 'style' attribute.

```
<h:message for="amt" style="color: red"/>
```

➔ One can display more than one error message using <h:messages> tag. It has 'layout' attribute which could be list or table. Default is list.

2.4. Changing Standard Error Message text

➔ Standard Conversion Error Messages are organized in key/value pairs. One example of standard error message is shown below. Key(Resource ID) and value(default value) are shown.



javax.faces.converter.NumberConverter.CURRENCY_detail	{2}: "{0}" could not be understood as a currency value. *Example: {1}*

Above key is for 'detail' message therefore value has 'Example:{1}' part.

➔ To replace standard message, one should set up a message bundle. Add the replacement message to the bundle with the appropriate key(shown in left in the pic above). Then set the base name of the bundle in configuration(faces-config.xml) file as shown below

```
<faces-config>
      <application>
            <message-bundle>com.corejsf.errmsgs</message-bundle>
      </application>
...
</faces-config>
```

➔ One need to only specify the messages that one need to override.

➔ Note that this message bundle is not same as <resource-bundle> which is used to map variables using value expression.

2.5. Using Custom Messages

➔ One can provide custom converter error message for a component using 'converterMessage' attribute.

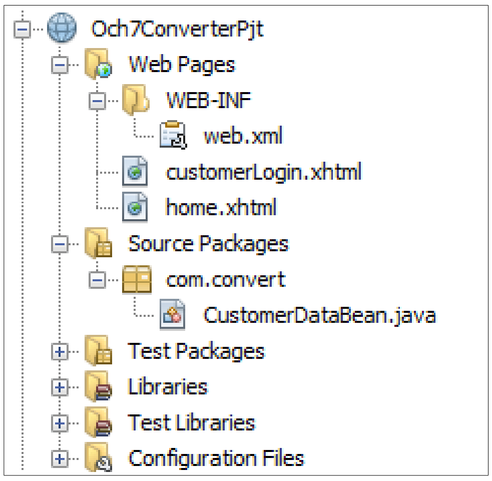
➔ Example

```
<h:outputLabel value="Amount :"/>
<h:inputText id="amt" value="#{customer.amount}" converterMessage="This is not valid
input">
<f:convertNumber minFractionDigits="2"/>
</h:inputText>
```

➔ When 'converterMessage' is used then detail or summary wont be displayed even 'label' attribute is present and

<h:message> tag is present. But it do take 'style' declared in <h:message>.

➔ Lets create a project 'Och7ConverterPjt' as shown below



```
Och7ConverterPjt
    Web Pages
        WEB-INF
            web.xml
        customerLogin.xhtml
        home.xhtml
    Source Packages
        com.convert
            CustomerDataBean.java
    Test Packages
    Libraries
    Test Libraries
    Configuration Files
```

CustomerBean.java

```
package com.convert;

import java.io.Serializable;
import java.util.Date;
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;

@Named(value = "customer")
@SessionScoped
public class CustomerDataBean implements Serializable {

    public CustomerDataBean() {
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public String getCcno() {
        return ccno;
    }

    public void setCcno(String ccno) {
        this.ccno = ccno;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    double amount;
    String ccno;
    Date date;
}
```

customerLogin.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Customer Login</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="3">

                <h:outputLabel value="Amount :"/>
                <h:inputText id="amount" label="This Is Invalid Input"
                value="#{customer.amount}">
                        <f:convertNumber minFractionDigits="2"/>
                </h:inputText>
                <h:message for="amount" showSummary="true" showDetail="false"
                style="color: red"/>

                <h:outputLabel value="Credit Card Number :"/>
                <h:inputText id="ccno" value="#{customer.ccno}"/>
                <h:message for="ccno" style="color: red"/>

                <h:outputLabel value="Valid Date :"/>
                <h:inputText id="date" value="#{customer.date}">
                    <f:convertDateTime pattern="mm/yyyy"/>
                </h:inputText>
                <h:message for="date" style="color: red"/>
            </h:panelGrid>
            <h:commandButton id="submit" value="submit" action="home"/>
        </h:form>
    </h:body>
</html>
```

home.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Home Page</title>
    </h:head>
    <h:body>

        <h:outputLabel value="Hello User"/><br/>
        <h:outputLabel value="Your Details"/><br/>
        <h:panelGrid columns="2">
        <h:outputLabel value="Amount :"/>
        <h:outputText value="#{customer.amount}"/>
        <ui:remove> <!-- <f:convertNumber type="currency"/>
        </h:outputText>--></ui:remove>
        <h:outputLabel value="Credit Card Number  :"/>
        <h:outputText value="#{customer.ccno}"/>
        <h:outputLabel value="Valid upto :"/>
        <h:outputText value="#{customer.date}"/><br/>

        </h:panelGrid>
    </h:body>
</html>
```

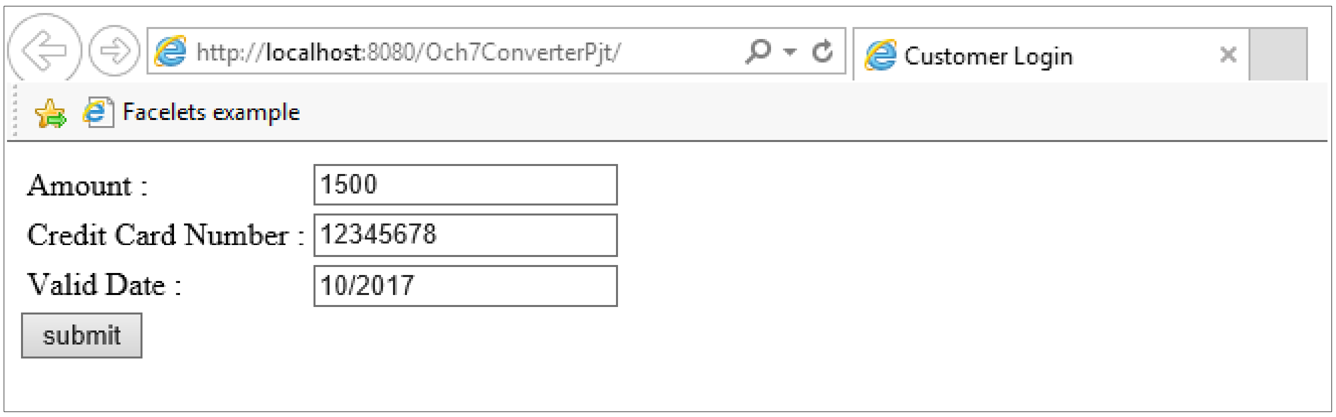
web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" 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-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <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>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/customerLogin.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

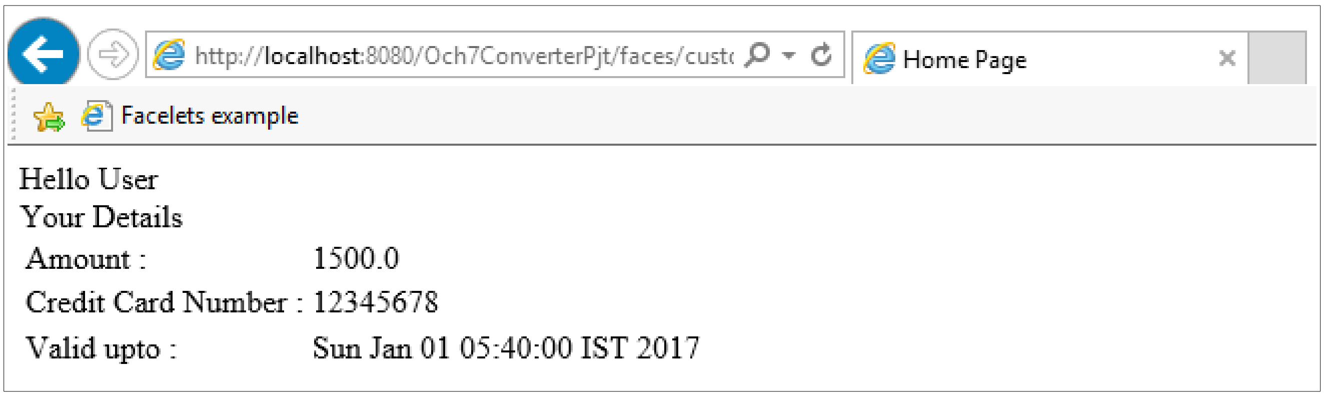
1. When valid input values are entered

customerLogin.xhtml



Amount : 1500
Credit Card Number : 12345678
Valid Date : 10/2017

home.xhtml

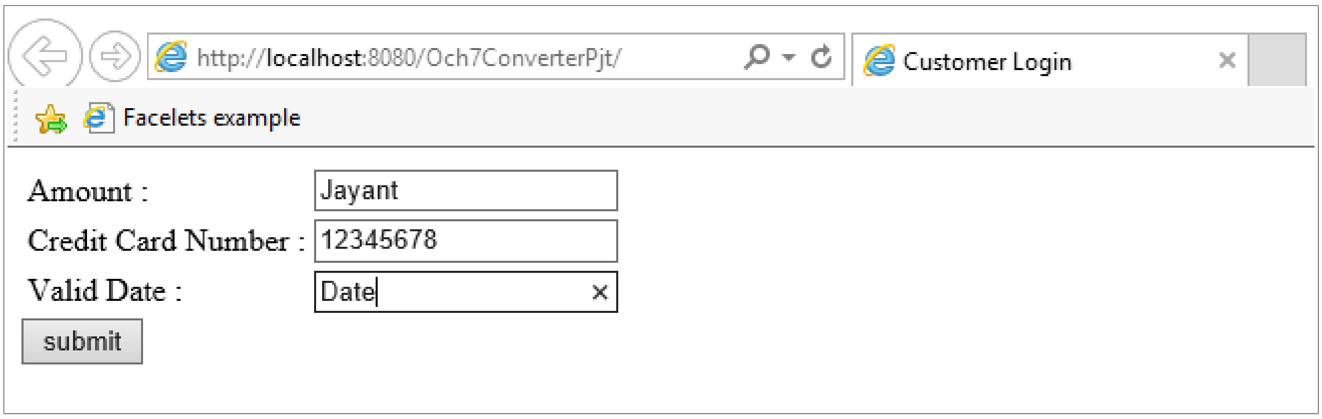

http://localhost:8080/Och7ConverterPjt/faces/custo
Home Page
Facelets example

Hello User
Your Details

Amount :	1500.0
Credit Card Number :	12345678
Valid upto :	Sun Jan 01 05:40:00 IST 2017

2. When Invalid values have been entered

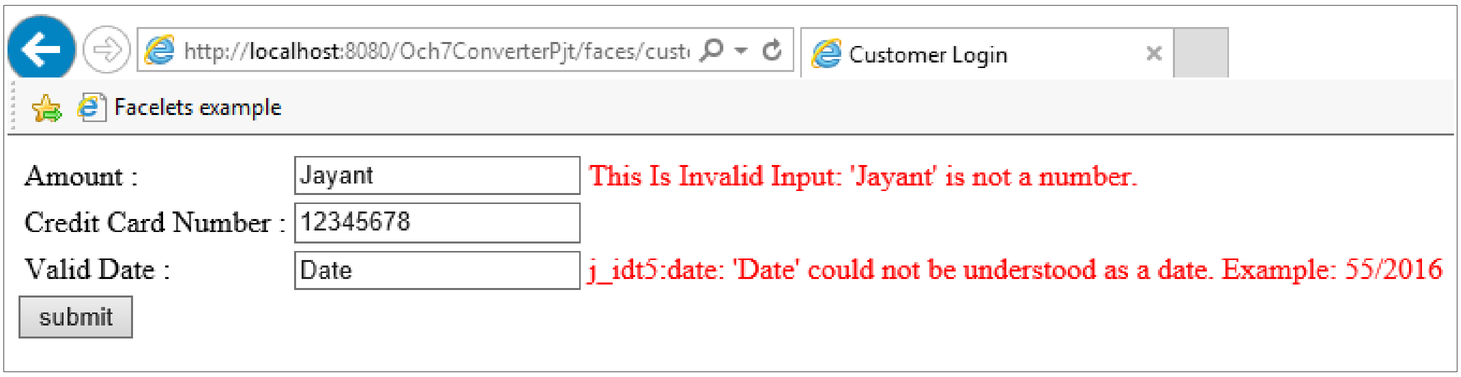
customerLogin..xhtml


http://localhost:8080/Och7ConverterPjt/
Customer Login
Facelets example

Amount :	Jayant
Credit Card Number :	12345678
Valid Date :	Date

submit

home.xhtml

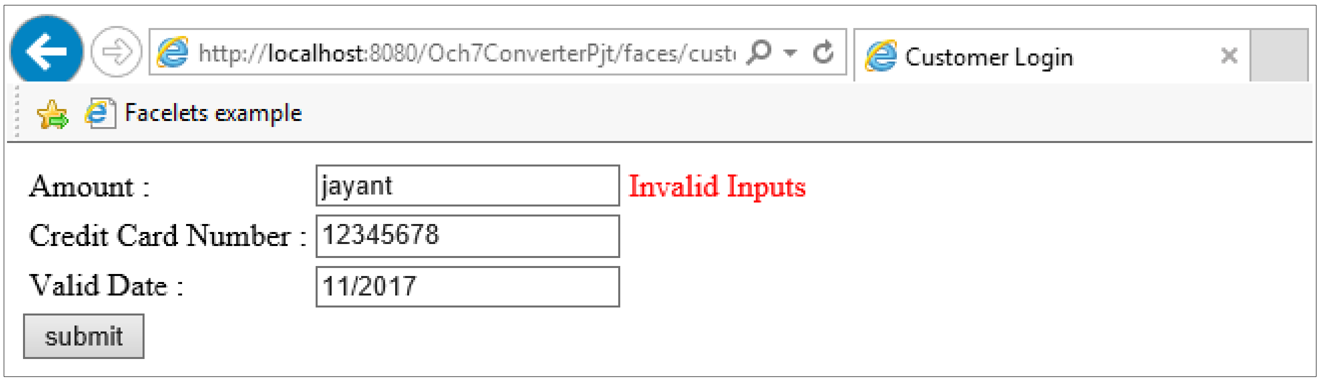

http://localhost:8080/Och7ConverterPjt/faces/custo
Customer Login
Facelets example

Amount :	Jayant	This Is Invalid Input: 'Jayant' is not a number.
Credit Card Number :	12345678	
Valid Date :	Date	j_idt5:date: 'Date' could not be understood as a date. Example: 55/2016

submit

3. If we use 'converterMessage'

```
<h:outputLabel value="Amount :"/>
      <h:inputText id="amount" label="This Is Invalid Input" value="#{customer.amount}"
converterMessage="Invalid Inputs">
            <f:convertNumber minFractionDigits="2"/>
      </h:inputText>
<h:message for="amount" showSummary="true" showDetail="false" style="color: red"/>
```


http://localhost:8080/Och7ConverterPjt/faces/cust
Customer Login
Facelets example

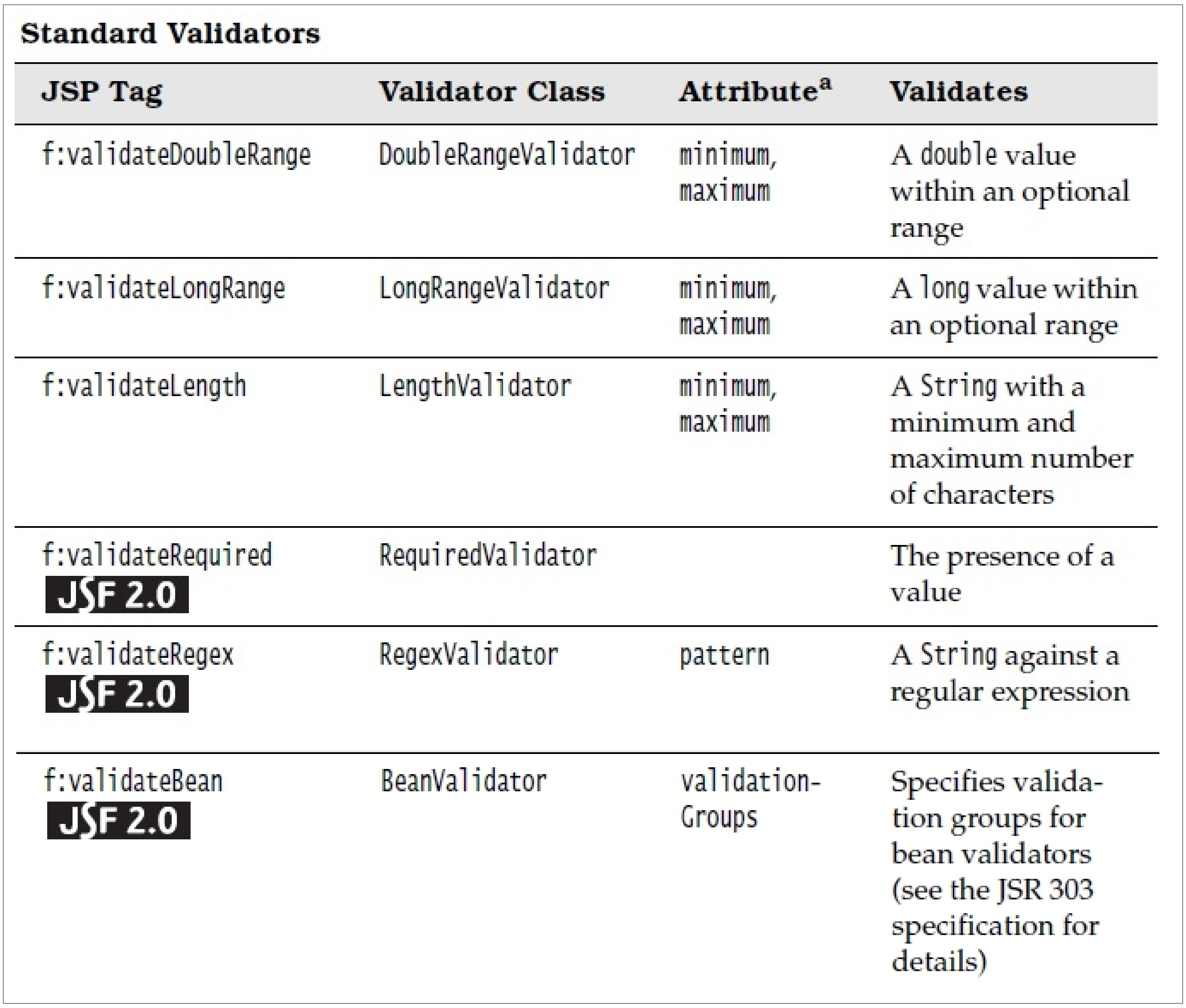
Amount :	jayant	Invalid Inputs
Credit Card Number :	12345678	
Valid Date :	11/2017	

submit

Note : Above we can see error message in 'converterMessage' has overridden all other messages but has taken red color from 'style' defined inside <h:message>.

3. Using Standard Validators

➔ The list of Standard Validators in JSF is shown below


Standard Validators

JSP Tag	Validator Class	Attribute[a]	Validates
f:validateDoubleRange	DoubleRangeValidator	minimum, maximum	A double value within an optional range
f:validateLongRange	LongRangeValidator	minimum, maximum	A long value within an optional range
f:validateLength	LengthValidator	minimum, maximum	A String with a minimum and maximum number of characters
f:validateRequired JSF 2.0	RequiredValidator		The presence of a value
f:validateRegex JSF 2.0	RegexValidator	pattern	A String against a regular expression
f:validateBean JSF 2.0	BeanValidator	validation-Groups	Specifies validation groups for bean validators (see the JSR 303 specification for details)

➔ JSF has build in mechanism which let one carry out the following validations.

a. Checking the length of a string

b. Checking limits for a numerical value (for example, > 0 or ≤ 100)

c. Checking against a regular expression (since JSF 2.0)

d. Checking that a value has been supplied

➔ Some of the validation examples are shown below

1. Validation for an Input field

```
<h:inputText id="ccno" value="#{customer.ccno}"/>
   <f:validateLength minimum="13"/>
</h:inputText>
```

The above code adds validator to a text field. When form is submitted validator makes sure that String contains minimum 13 characters. Otherwise it will generate error messages associated with the guilty component. The message text later can be displayed using `<h:message>` and `<h:messages>` tags.

2. Validating a long range

```
<h:inputText id="amount" value="#{customer.amount}">
   <f:validateLongRange minimum="10" maximum="10000"/>
</h:inputText>
```

The validator checks that supplied value is $\geq$ 10 and $\leq$ 10000.

3. To check that a value is supplied or not

```
<h:inputText id="date" value="#{customer.date}">
   <f:validateRequired/>
</h:inputText>
```

'`ValidateRequired`' simply set '`required`' attribute of the enclosing component to true. Hence alternatively one can also use

```
<h:inputText id="date" value="#{customer.date}" required="true"/>
```

➔ An alternative way to attach validation is through `<f:validator>` tag. Here one needs to supply Validator ID and parameters as shown below

```
<h:inputText id="ccno" value="#{customer.ccno}"/>
   <f:validator validatorId="javax.faces.validator.LengthValidator">
   <f:attribute name="ccno" value="13"/>
   </f:validator>
</h:inputText>
```

3.1. Displaying Validation Errors

➔ Displaying message for validation is similar to the displaying conversion errors. A message is added to the component that failed validation, the component is invalidated and the current page is redisplayed immediately after 'Process Validation' phase has completed.

➔ One can use `<h:message>` and `<h:messages>` tags to display messages.

➔ To override standard validation messages one can create message bundle and follow the process defined during standard conversion error overriding.

3.2. Bypassing Validation

➔ Sometimes its necessary to bypass validation process. For example suppose cancel button on the page has 'required' field. So when user clicks on cancel button same page will be redisplayed with the validation error messages.

➔ To bypass it if command has 'immediate' attribute set then command is executed during 'Apply Request Value' phase.

```
<h:commandButton value="Cancel" action="cancel" immediate="true"/>
```

➔ Ex. Lets change customerLogin.xhtml page to as shown below

customerLogin.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Customer Login</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="3">

                <h:outputLabel value="Amount :"/>
                <h:inputText id="amount" label="This Is Invalid Input"
                value="#{customer.amount}" required="true">
                        <f:convertNumber minFractionDigits="2"/>
                <f:validateDoubleRange minimum="10" maximum="10000"/>
                </h:inputText>
                <h:message for="amount" showSummary="true" showDetail="false"
                style="color: red"/>

                <h:outputLabel value="Credit Card Number :"/>
                <h:inputText id="ccno" value="#{customer.ccno}" required="true"
                                      requiredMessage="Credit Card No is Required">
                  <f:validateLength minimum="8"/>
                </h:inputText>
                <h:message for="ccno" style="color: red"/>

                <h:outputLabel value="Valid Date :"/>
                <h:inputText id="date" value="#{customer.date}">
                    <f:convertDateTime pattern="mm/yyyy"/>
                </h:inputText>
                <h:message for="date" style="color: red"/>
            </h:panelGrid>
            <h:commandButton id="submit" value="submit" action="home"/>
            <h:commandButton value="Cancel" action="canceled" immediate="true"/>
        </h:form>
    </h:body>
</html>
```

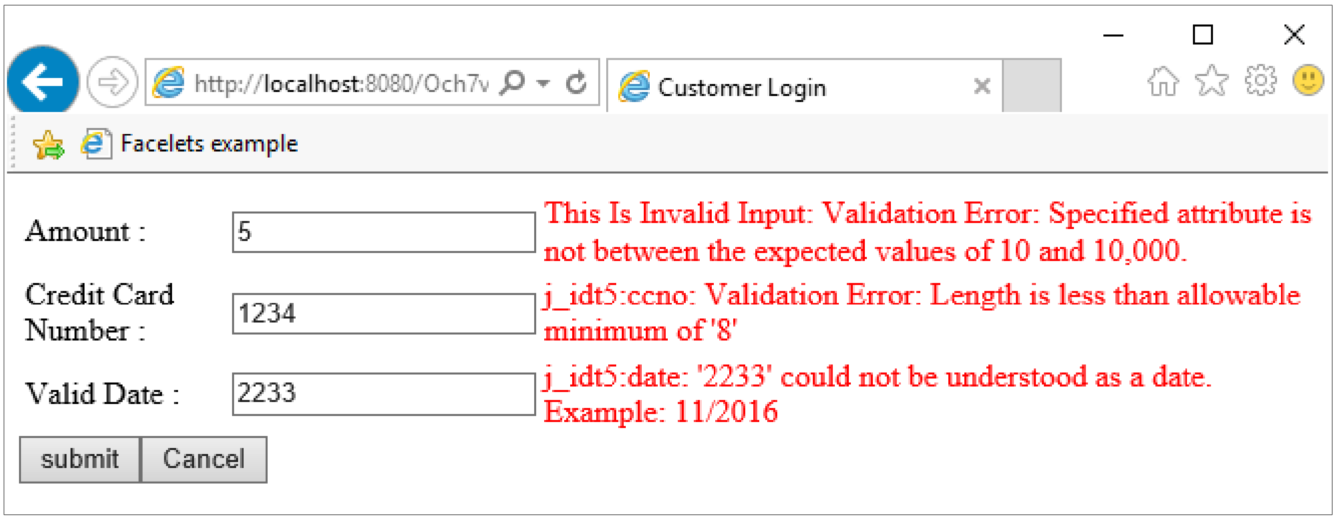
canceled.xhtml

```
<?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:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Canceled Page</title>
  </h:head>
  <h:body>
    <h:form>
      The transaction has been canceled.<br/>
      <h:commandButton value="Back" action="customerLogin"/>
    </h:form>
  </h:body>
</html>
```

Output:

1. When invalid input has been entered

customerLogin.xhtml



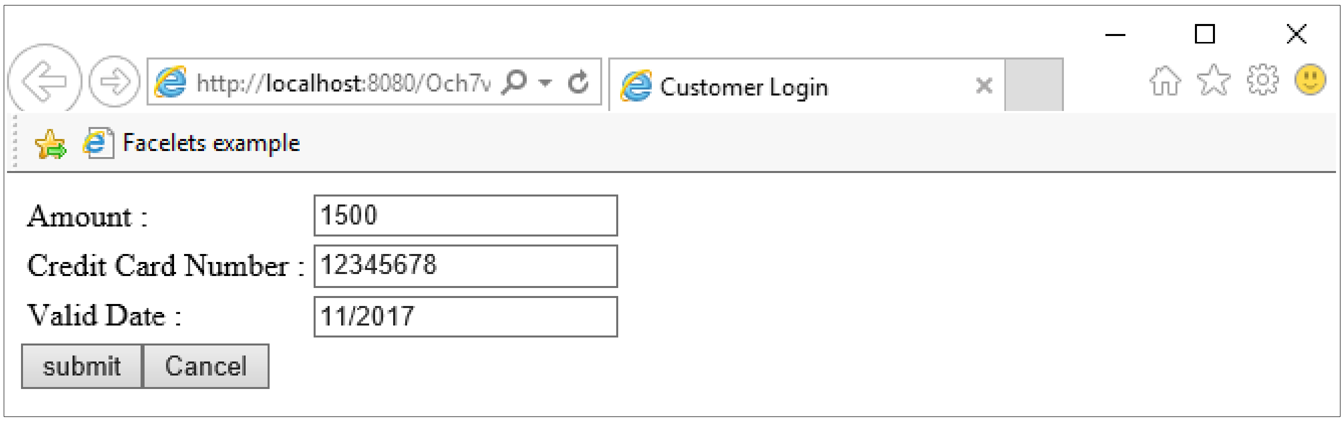
Amount : 5 This Is Invalid Input: Validation Error: Specified attribute is not between the expected values of 10 and 10,000.

Credit Card Number : 1234 j_idt5:ccno: Validation Error: Length is less than allowable minimum of '8'

Valid Date : 2233 j_idt5:date: '2233' could not be understood as a date. Example: 11/2016

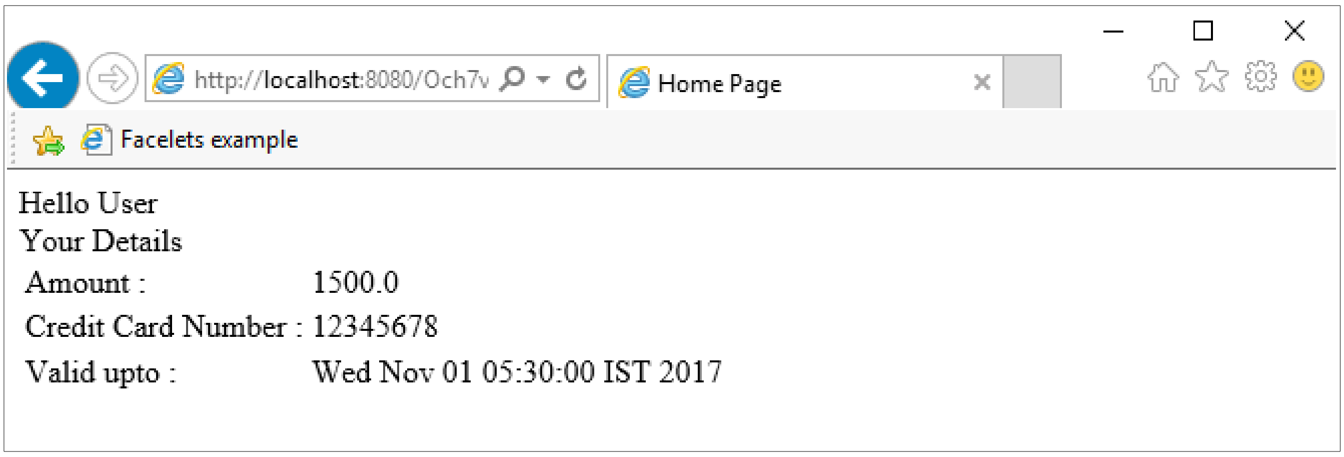
2. When Valid input has been entered

customerLogin.xhtml



Amount :	1500
Credit Card Number :	12345678
Valid Date :	11/2017

home.xhtml



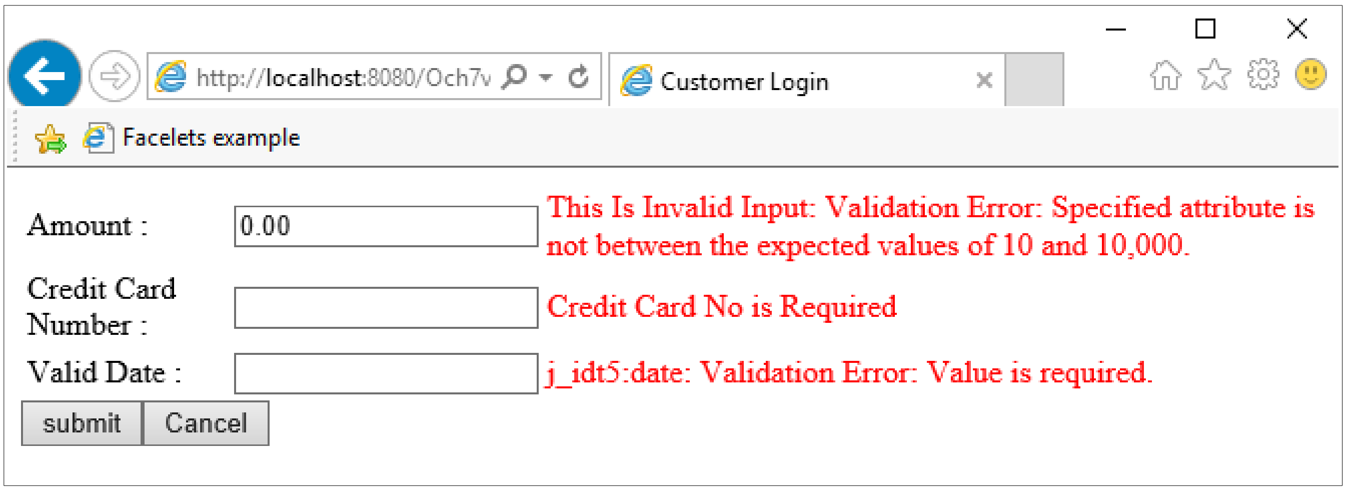
Hello User
Your Details

Amount :	1500.0
Credit Card Number :	12345678
Valid upto :	Wed Nov 01 05:30:00 IST 2017

3. 'Immediate' attribute.

a. When 'immediate' attribute is not present and cancel button is pressed.

customerLogin.xhtml



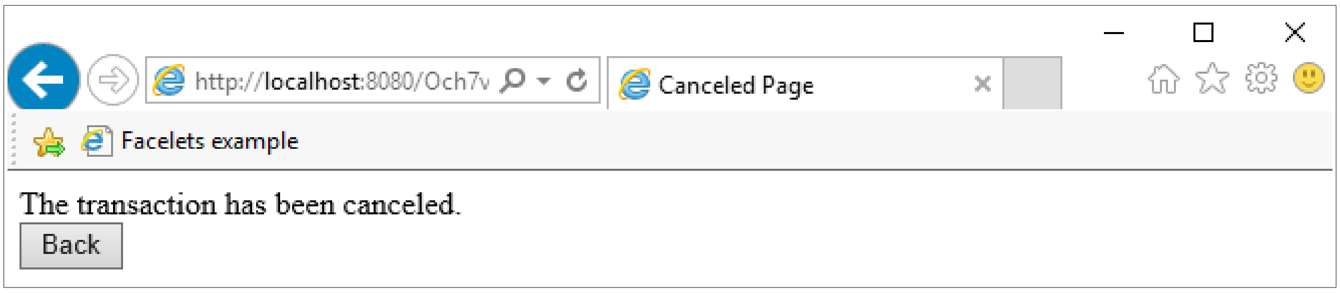
Amount : 0.00 This Is Invalid Input: Validation Error: Specified attribute is not between the expected values of 10 and 10,000.

Credit Card Number : Credit Card No is Required

Valid Date : j_idt5:date: Validation Error: Value is required.

b. When 'immediate' attribute is set and cancel button is pressed.

canceled.xhtml



The transaction has been canceled.

4. **Bean Validation**

➔ JSF 2.0 integrates with bean validation framework, a general framework for speficifying validation constraints.

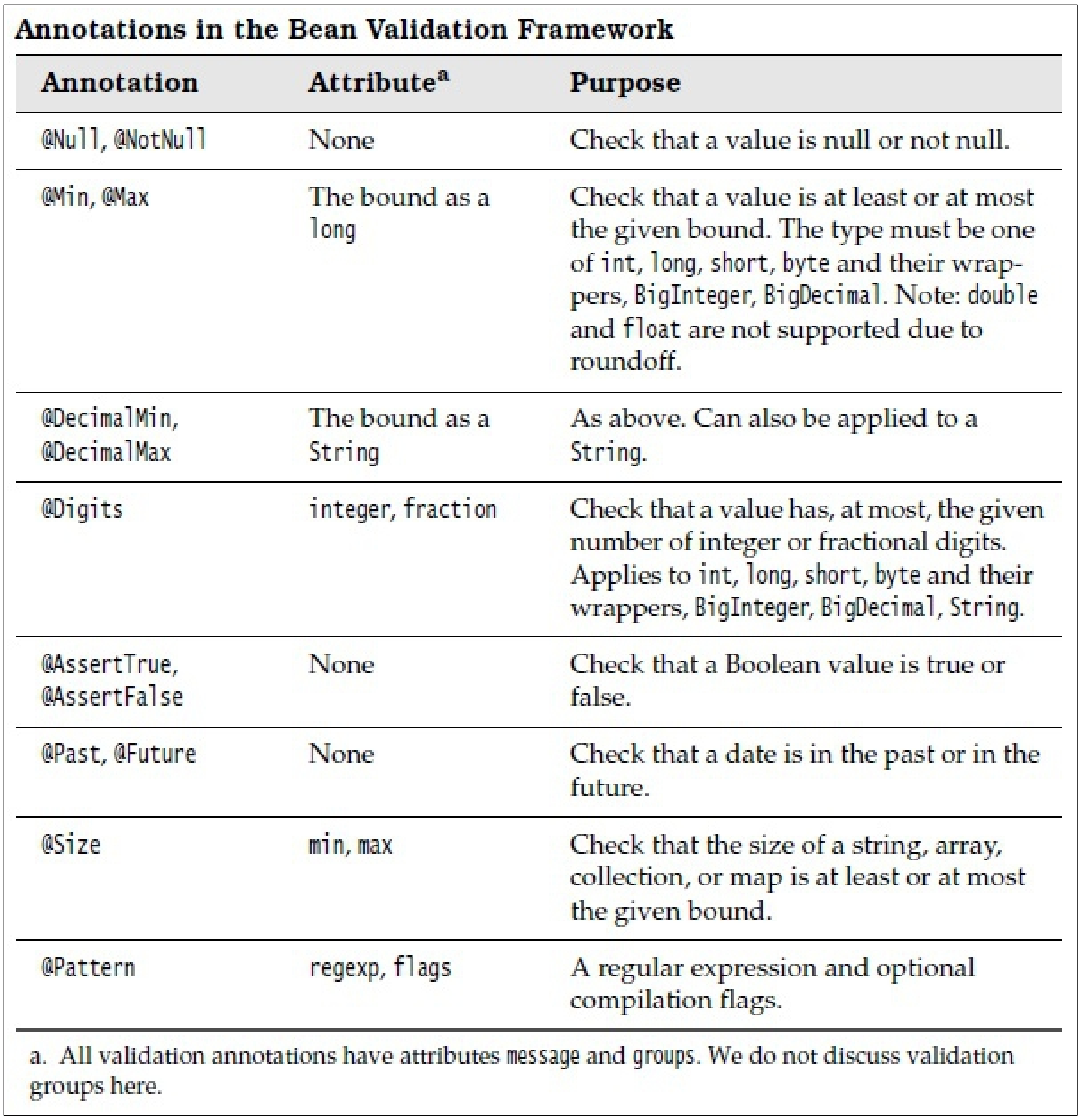
➔ Bean validation framework has tremendous advantage over page-level validation. Suppose a web application updates a bean in several pages. Then one don't need to add validation rules in each page, they will be handled in bean class.

➔ Validations are attached to fields or property getters of a java class.

➔ Ex

```
public class CustomerBean
{
@Size(min=13) private String ccno;
@Future public data getDate(){...}
….
}
```

➔ The list of all the annotations for bean validation is shown below



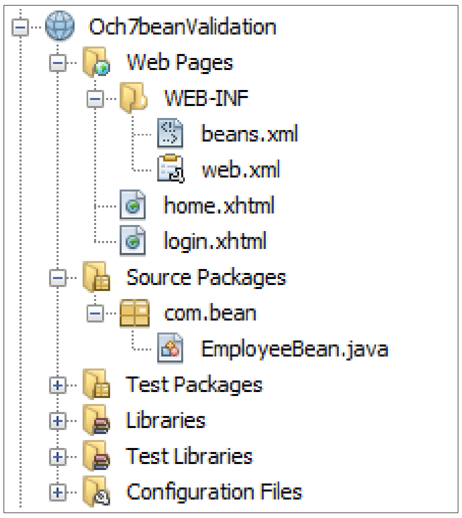
Annotations in the Bean Validation Framework

Annotation	Attribute[a]	Purpose
`@Null`, `@NotNull`	None	Check that a value is null or not null.
`@Min`, `@Max`	The bound as a `long`	Check that a value is at least or at most the given bound. The type must be one of `int`, `long`, `short`, `byte` and their wrappers, `BigInteger`, `BigDecimal`. Note: `double` and `float` are not supported due to roundoff.
`@DecimalMin`, `@DecimalMax`	The bound as a `String`	As above. Can also be applied to a `String`.
`@Digits`	`integer`, `fraction`	Check that a value has, at most, the given number of integer or fractional digits. Applies to `int`, `long`, `short`, `byte` and their wrappers, `BigInteger`, `BigDecimal`, `String`.
`@AssertTrue`, `@AssertFalse`	None	Check that a Boolean value is true or false.
`@Past`, `@Future`	None	Check that a date is in the past or in the future.
`@Size`	`min`, `max`	Check that the size of a string, array, collection, or map is at least or at most the given bound.
`@Pattern`	`regexp`, `flags`	A regular expression and optional compilation flags.

a. All validation annotations have attributes `message` and `groups`. We do not discuss validation groups here.

➔ EX

Lets have directory for project 'Och7beanValidation' as shown below



EmployeeBean.java

```java
package com.bean;

import java.io.Serializable;
import javax.inject.Named;
import javax.enterprise.context.Dependent;
import javax.enterprise.context.SessionScoped;
import javax.validation.constraints.Digits;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.Size;

@Named(value = "employee")
@SessionScoped
public class EmployeeBean implements Serializable {

    public EmployeeBean() {
    }
    @Size(max=8,message="Name shouldn't exceed 8 characters")
    String name;
    @Digits(integer=4,fraction=0)
    int empid;
    @Min(25) @Max(50)
    int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getEmpid() {
        return empid;
    }

    public void setEmpid(int empid) {
        this.empid = empid;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

login.xhtml

```xml
<?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>Login Page</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel value="Employee Name :"/>
                <h:inputText id="name" value="#{employee.name}" required="true"/>

                <h:outputLabel value="Employee Age :"/>
                <h:inputText id="age" value="#{employee.age}"/>

                <h:outputLabel value="Emp ID :"/>
                <h:inputText id="empid" value="#{employee.empid}"/>
            </h:panelGrid>
            <h:commandButton id="submit" value="submit" action="home"/>

        </h:form>
    </h:body>
</html>
```

home.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>Home Page</title>
    </h:head>
    <h:body>
        <h1>Employee Details</h1>
        <h:panelGrid columns="2">
           <h:outputLabel value="Employee Name : "/>
           <h:outputText value="#{employee.name}"/>
           <h:outputLabel value="Employee Age : "/>
           <h:outputText value="#{employee.age}"/>
           <h:outputLabel value="Employee ID : "/>
           <h:outputText value="#{employee.empid}"/>
        </h:panelGrid>

    </h:body>
</html>
```

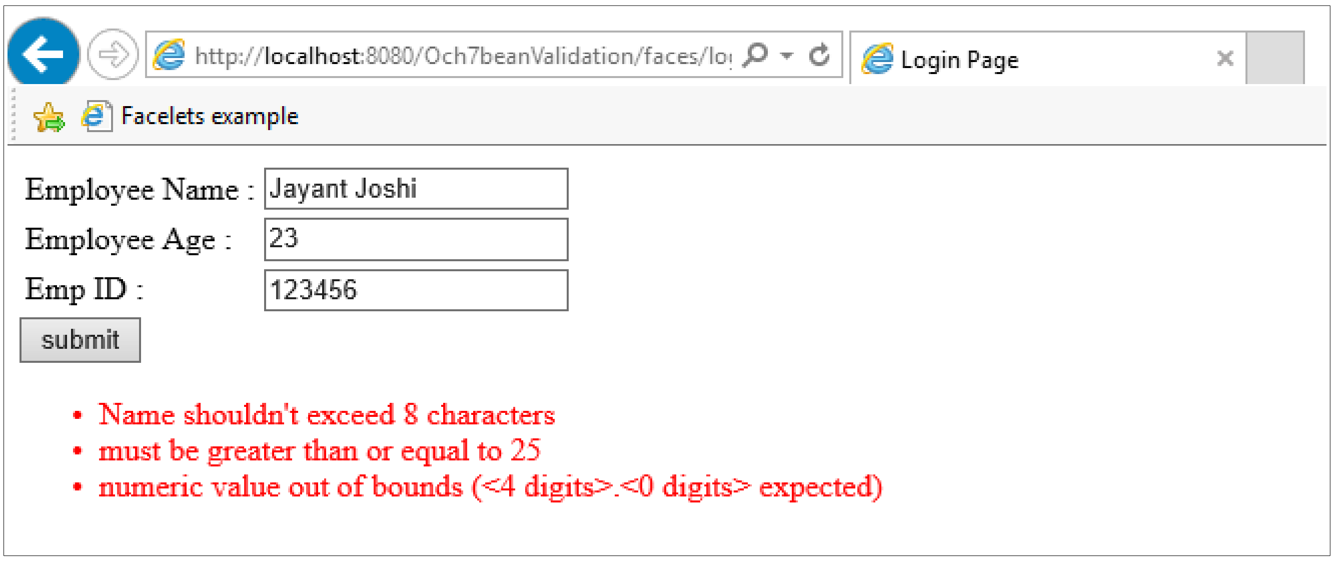
web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" 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-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <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>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/login.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

1. Invalid inputs

login.xhtml

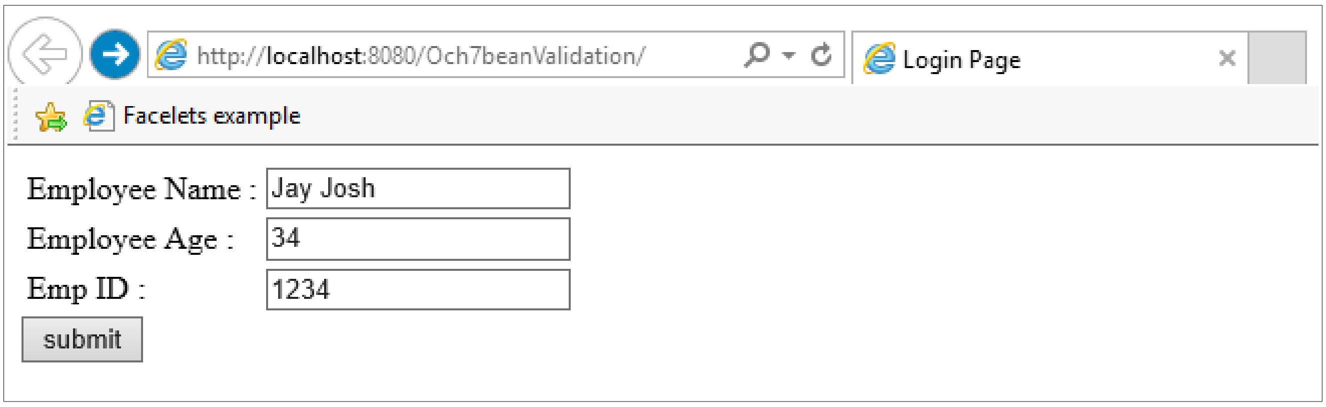


Employee Name : Jayant Joshi
Employee Age : 23
Emp ID : 123456

- Name shouldn't exceed 8 characters
- must be greater than or equal to 25
- numeric value out of bounds (<4 digits>.<0 digits> expected)

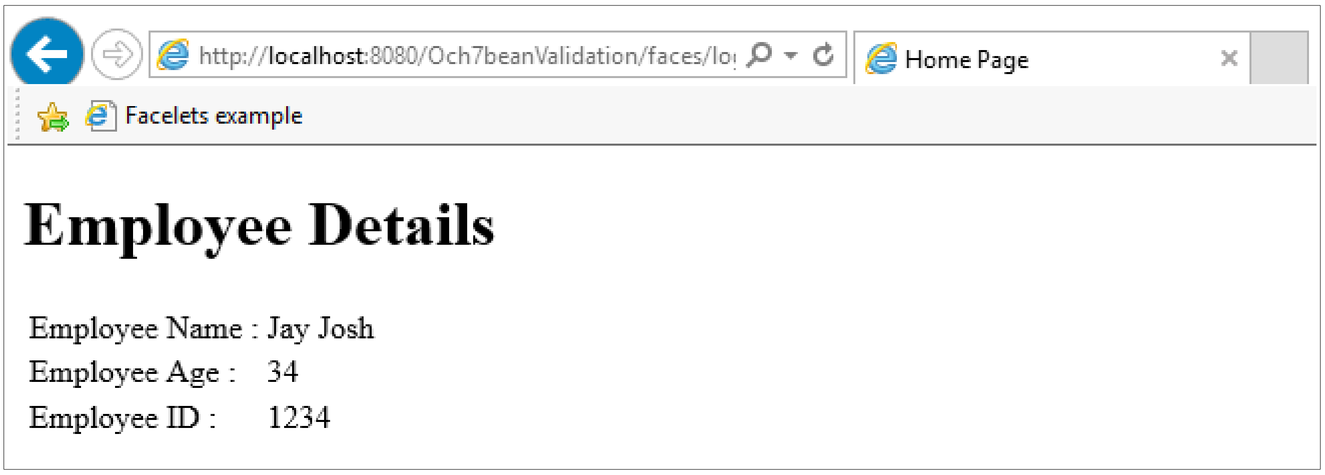
2. Valid Inputs

login.xhtml



Employee Name : Jay Josh
Employee Age : 34
Emp ID : 1234

home.xhtml



Employee Details

Employee Name : Jay Josh
Employee Age : 34
Employee ID : 1234

5. <u>Implementing Custom Converter Classes</u>

➔ To implement custom converter one should implement interface '`Converter`'.

➔ `Converter` interface has two methods

```
Object getAsObject(FacesContext context, UIComponent component, String newValue)
String getAsString(FacesContext context, UIComponent component, Object value)
```

➔ Method `getAsObject()` convert String (`newValue`, which comes from `client/browser`) into object of particular type. It throws `ConverterException` if conversion fails. Method `getAsString()` converts `Object(value)` from object to String and throws `ConversionException` if conversion fails.

➔ Converter can be specify using annotation `@FacesConverter` in two ways

i. <u>Using '`converterId`'</u>

In bean class use annotation as shown below

```
@FacesConverter("com.custom.amountConverter")
public class AmountConverter implements Converter {
```

Later it will be used in view as

```
<h:outputLabel value="Amount :"/>
<h:inputText id="amount" value="#{customer.amount}">
   <f:converter converterId="com.custom.amountConverter"/>
 </h:inputText>
 <h:message for="amount"/>
```

2. <u>Using '`forClass`'</u>

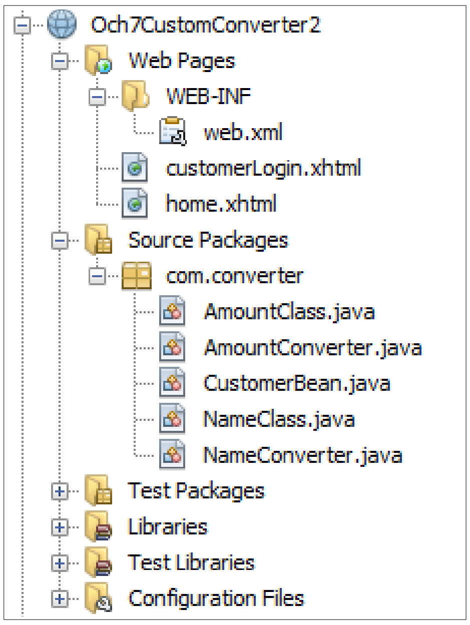
```
@FacesConverter(forClass=NameClass.class)
public class NameConverter implements Converter {
```

With this no need to mention converter any longer. It will be automatically used whenever a value reference has the type '`NameClass`'.

```
 <h:outputLabel value="Name :"/>
   <h:inputText id="name" value="#{customer.nc}"/>
<h:message for="name"/>
```

➔ Example

Lets create a directory 'Och7CustomConverter2'



```
Och7CustomConverter2
  Web Pages
    WEB-INF
      web.xml
    customerLogin.xhtml
    home.xhtml
  Source Packages
    com.converter
      AmountClass.java
      AmountConverter.java
      CustomerBean.java
      NameClass.java
      NameConverter.java
  Test Packages
  Libraries
  Test Libraries
  Configuration Files
```

customerBean.java

```java
package com.converter;

import java.io.Serializable;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.Size;

@Named("customer")
@SessionScoped
public class CustomerBean implements Serializable {

    private NameClass nc = new NameClass("");
    private String amount;
    private String ccno;

    public NameClass getNc() {
        return nc;
    }

    public void setNc(NameClass nc) {
        this.nc = nc;
    }

    public String getAmount() {
        return amount;
    }

    public void setAmount(String amount) {
        this.amount = amount;
    }

    public String getCcno() {
        return ccno;
    }

    public void setCcno(String ccno) {
        this.ccno = ccno;
    }
}
```

NameClass.java

```java
package com.converter;

import java.io.Serializable;

public class NameClass implements Serializable {

    String name;

    public NameClass(String name)
    {
        this.name = name;
    }

    @Override
    public String toString()
    {
        return name;
    }
}
```

NameConverter.java

```
package com.converter;

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;

@FacesConverter(forClass=NameClass.class)
public class NameConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext context,
            UIComponent component, String value) throws ConverterException{

        String str="";
        char [] valArr = value.toCharArray();
        for(int i=0; i<value.length(); i++)
        {
         str = str+valArr[i]+" ";
        }

        NameClass nc = new NameClass(str);
        return nc;
    }

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

        String str = value.toString();
        return str;
    }
}
```

AmountClass.java

```
package com.converter;

import java.io.Serializable;
import javax.validation.constraints.Size;

public class AmountClass implements Serializable{


    private String amount;

    public AmountClass(String amount)
    {
        this.amount=amount;
    }

    public String toString()
    {
        return amount;
    }

}
```

AmountConverter.java

```
package com.converter;

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;

@FacesConverter("com.custom.amountConverter")
```

```java
public class AmountConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext context,
            UIComponent component, String value) throws ConverterException {

        String str = "";
        str = "â,¹"+value;

        AmountClass ac = new AmountClass(str);
        return ac;

    }

    @Override
    public String getAsString(FacesContext context,
            UIComponent component, Object value) throws ConverterException {
         String str = value.toString();
         return str;
    }
}
```

Customer.xhtml

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Customer Login</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="3">
                <h:outputText value="Name :"/>
                <h:inputText id="name" value="#{customer.nc}"/>
                <h:message for="name" style="color: red"/>

                 <h:outputText value="Amount :"/>
                 <h:inputText id="amount" value="#{customer.amount}">
                    <f:converter converterId="com.custom.amountConverter"/>
                </h:inputText>
                 <h:message for="amount" style="color: red"/>

                 <h:outputText value="Credit Card Number:"/>
                 <h:inputText id="ccno" value="#{customer.ccno}"/>
                 <h:message for="ccno" style="color: red"/>
            </h:panelGrid>

            <h:commandButton id="submit" value="submit" action="home"/>
        </h:form>
    </h:body>
</html>
```

home.xhtml

```xml
<?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>Home Page</title>
    </h:head>
    <h1>Customer Details</h1>
    <h:body>
        <h:panelGrid columns="2">
          <h:outputText value="Name: "/>
```

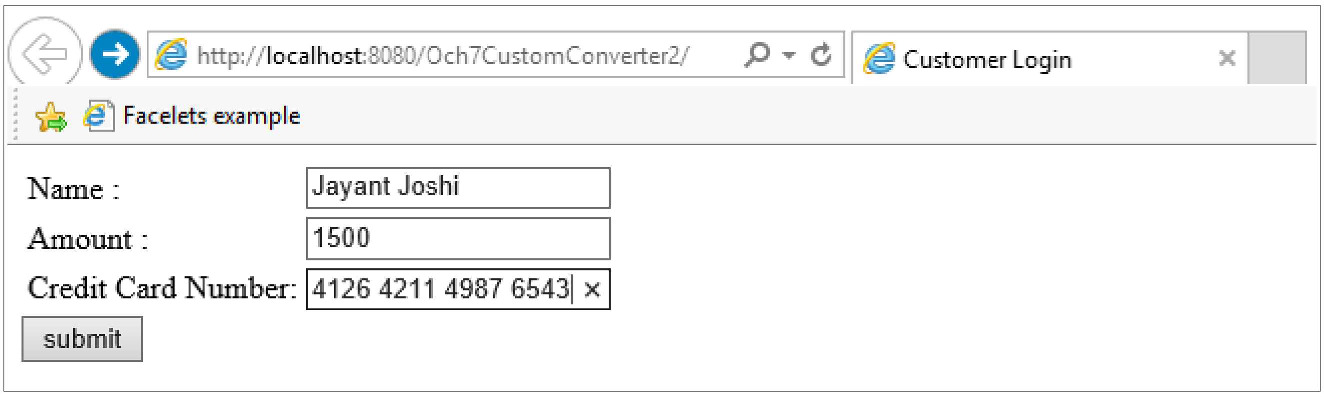
```
            <h:outputText value="#{customer.nc}"/>
            <h:outputText value="Amount Entered: "/>
            <h:outputText value="#{customer.amount}"/>
            <h:outputText value="Credit Card Number: "/>
            <h:outputText value="#{customer.ccno}"/>
        </h:panelGrid>

    </h:body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" 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-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <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>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/customerLogin.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

customerLogin.xhtml

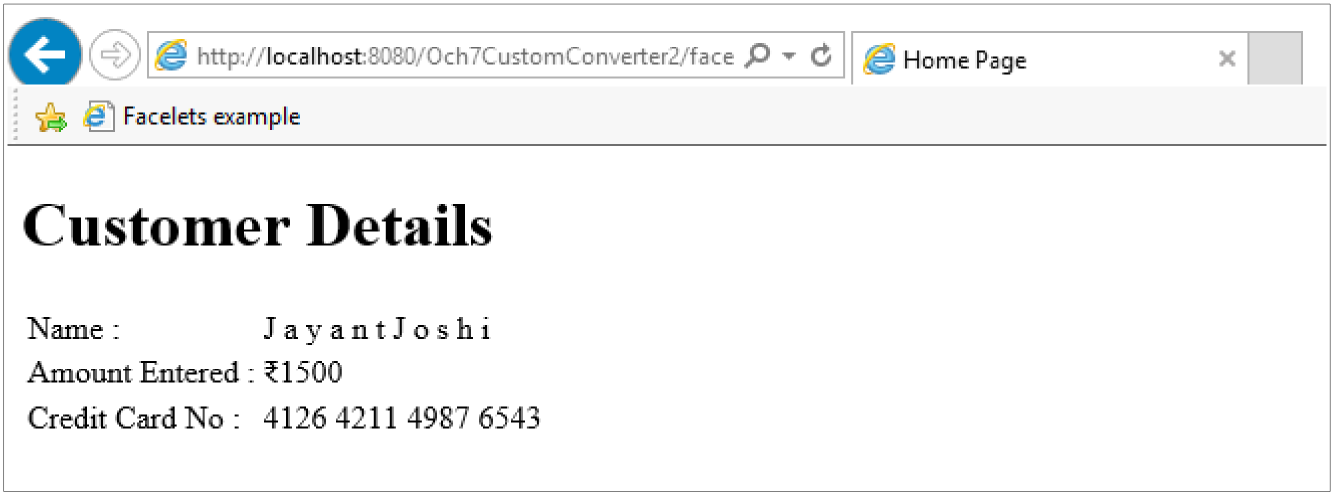

http://localhost:8080/Och7CustomConverter2/

Name : Jayant Joshi

Amount : 1500

Credit Card Number: 4126 4211 4987 6543

home.xhtml



Customer Details

Name : J a y a n t J o s h i
Amount Entered : ₹1500
Credit Card No : 4126 4211 4987 6543

5.1. Supplying Attributes to Converter

One can supply attributes to a converter for example in above example if we want credit card number to be separated by '-' then it can be supplied to converter using

```
<h:outputText value="Credit Card Number:"/>
      <h:inputText id="ccno" value="#{customer.ccno}"/>
      <f:attribute name="separator" value="-"/>
<h:message for="ccno" style="color: red"/>
```

Inside converter this attribute will be retrieved using

```
separator = (String) component.getAttributes().get("separator");
```

6. Implementing Custom Validator Classes

➔ Implementing Custom Validator is similar to implementing Custom Converters.

➔ To implement Custom Validator one should implement interface 'Validator' and it will throw 'ValidatorException' if validation fails.

➔ Validator interface has only one method as shown below

```
void validate(FacesContext context, UIComponent component, Object value)
```

➔ One should register custom validator similar to custom validator

```
@FacesValidator("com.converter.Name")
public class NameValidator implements Validator {
```

And then used in view page as

```
<h:outputText value="Name :"/>
                <h:inputText id="name" value="#{customer.nc}">
                    <f:validator validatorId="com.converter.Name"/>
                </h:inputText>
<h:message for="name" style="color: red"/>
```

➔ 'ValidatorException' class has following Constructors.

```
ValidatorException(java.util.Collection<FacesMessage> messages)
ValidatorException(java.util.Collection<FacesMessage> messages, java.lang.Throwable cause)
ValidatorException(FacesMessage message)
ValidatorException(FacesMessage message, java.lang.Throwable cause)
```

➔ Examples for generating 'Validation Errors' messages.

1) By directly providing the error message

```
FacesMessage message = new FacesMessage("Name exceeded 10 characters");
message.setSeverity(FacesMessage.SEVERITY_ERROR);
```

```
throw new ValidatorException(message);
```

2) By getting message through message bundle

```
FacesMessage message = com.converter.util.Messages.getMessage("com.converter.messages",
"nameExceeded", null);
message.setSeverity(FacesMessage.SEVERITY_ERROR);
throw new ValidatorException(message);
```

Above `com.converter.util` is a message which contains `Message.java` class which has `getMessage()` method which gets the method from '`messages.properties`' bundle present in `com.converter` package.

➔ Examples

Lets create '`NameValidator`' and '`AmountValidator`' for the project '`Och7CustomConverter2`' as shown below

NameValidator.java

```
package com.converter;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;
import javax.validation.ValidationException;

@FacesValidator("com.converter.nameValidator")
public class NameValidator implements Validator {

    @Override
    public void validate(FacesContext context,
            UIComponent component, Object value)   {

        String str = value.toString();

         if(str.length()>8)
         {
             FacesMessage message = new FacesMessage("Name exceeded 10 characters");
             message.setSeverity(FacesMessage.SEVERITY_ERROR);
             throw new ValidatorException(message);
         }
    }
}
```

AmountValidator.java

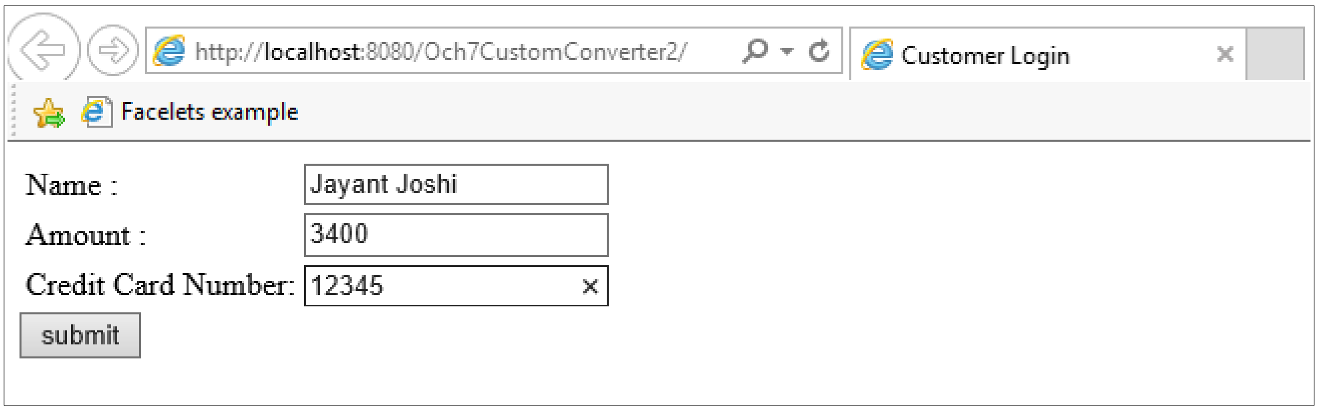
```
package com.converter;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

@FacesValidator("com.convert.amountValidator")
public class AmountValidator implements Validator {

    @Override
    public void validate(FacesContext context,
            UIComponent component, Object value) throws ValidatorException {

        String str = value.toString();
        String str1 = "";
        char [] strArr = str.toCharArray();
        for(int i=1; i<strArr.length; i++)
        {
            str1=str1+strArr[i];
            str=str1;
        }
        double amt = Double.parseDouble(str);
        if(amt>2000)
        {
            FacesMessage message = new FacesMessage("Amount Should be < 2000");
            message.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ValidatorException(message);
        } } }
```

customerLogin.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Customer Login</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="3">
                <h:outputText value="Name :"/>
                <h:inputText id="name" value="#{customer.nc}">
                    <f:validator validatorId="com.converter.nameValidator"/>
                </h:inputText>
                <h:message for="name" style="color: red"/>

                 <h:outputText value="Amount :"/>
                 <h:inputText id="amount" value="#{customer.amount}">
                    <f:converter converterId="com.custom.amountConverter"/>
                    <f:validator validatorId="com.convert.amountValidator"/>
                 </h:inputText>
                 <h:message for="amount" style="color: red"/>

                 <h:outputText value="Credit Card Number:"/>
                 <h:inputText id="ccno" value="#{customer.ccno}"/>
                 <h:message for="ccno" style="color: red"/>
            </h:panelGrid>

            <h:commandButton id="submit" value="submit" action="home"/>
        </h:form>
    </h:body>
</html>
```

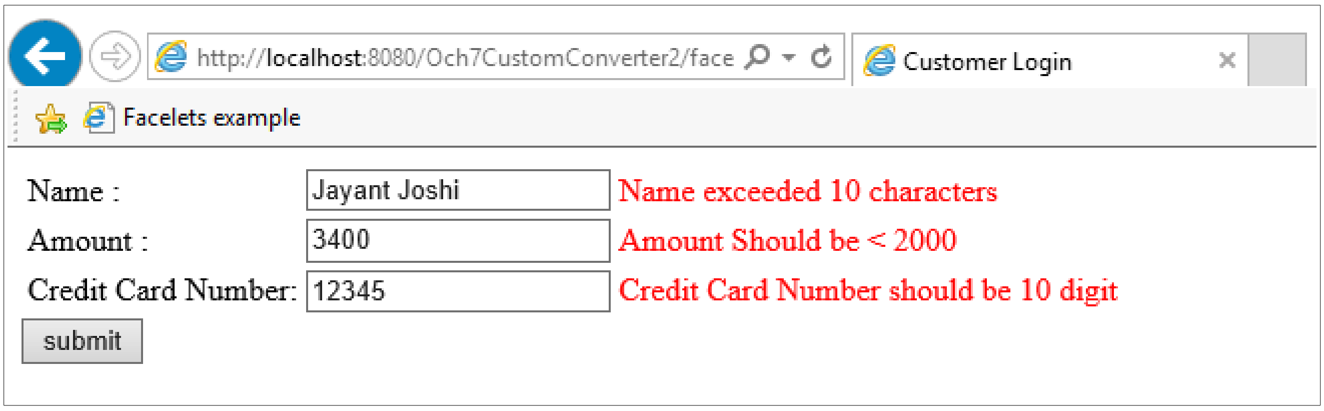
Output

customerLogin.xhtml



Name : Jayant Joshi
Amount : 3400
Credit Card Number: 12345

home.xhtml



Name : Jayant Joshi Name exceeded 10 characters
Amount : 3400 Amount Should be < 2000
Credit Card Number: 12345 Credit Card Number should be 10 digit

Note: Annotation

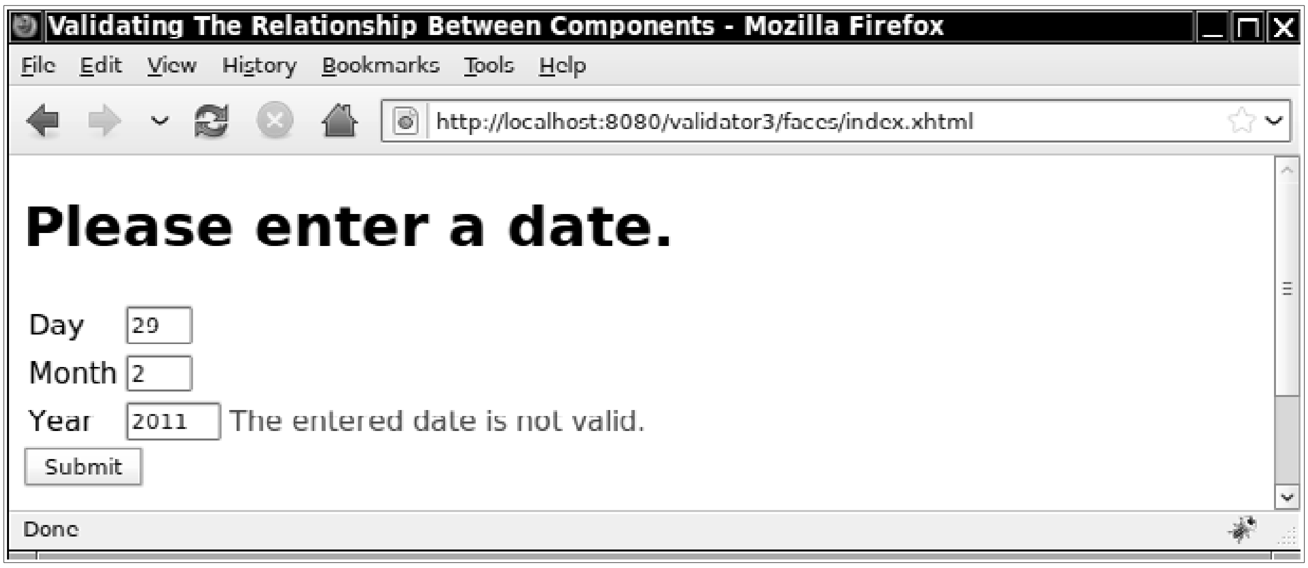
```
@Size(min=10,max=10,message="Credit Card Number should be 10 digit")
private String ccno;
```

has been added to '`CustomerBean.java`' class

➔ One can do validation with bean methods too using method expression. Its advantage is validation method can access instance variable of the class. But the disadantage is such validator cannot be moved to other application for re-use.

6.1. Validating relationships between different components

➔ Validation mechanism could be used only for one component. If there are related components example 'Date' where three different input filed has been given for data, month and year as shown below


Validating The Relationship Between Components - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/validator3/faces/index.xhtml

Please enter a date.

Day 29
Month 2
Year 2011 The entered date is not valid.

➔ In such cases we don't want any field don't send wrong value to the web app example feb 30.

➔ To resolve this we can use validator for last component and could also validate other related component thereusing their component IDs . Example is shown below

```
public class BackingBean {

...
public void validateDate(FacesContext context, UIComponent component,
Object value) {
UIInput dayInput = (UIInput) component.findComponent("day");
UIInput monthInput = (UIInput) component.findComponent("month");
int d = ((Integer) dayInput.getLocalValue()).intValue();
int m = ((Integer) monthInput.getLocalValue()).intValue();
int y = ((Integer) value).intValue();
if (!isValidDate(d, m, y)) {
FacesMessage message = ...;
throw new ValidatorException(message);
}
}
...
}
```

➔ An alternative approach is to attach the validator to a *hidden input field* that comes after all other fields on the form:

```
<h:inputHidden id="datecheck" validator="#{bb.validateDate}" value="needed"/>
```

The hidden field is rendered as a hidden HTML input field. When the field value is posted back, the validator kicks in. (It is essential that you supply some field value. Otherwise, the component value is never updated.) With this approach, the validation function is more symmetrical since all other form components already have their local values set.