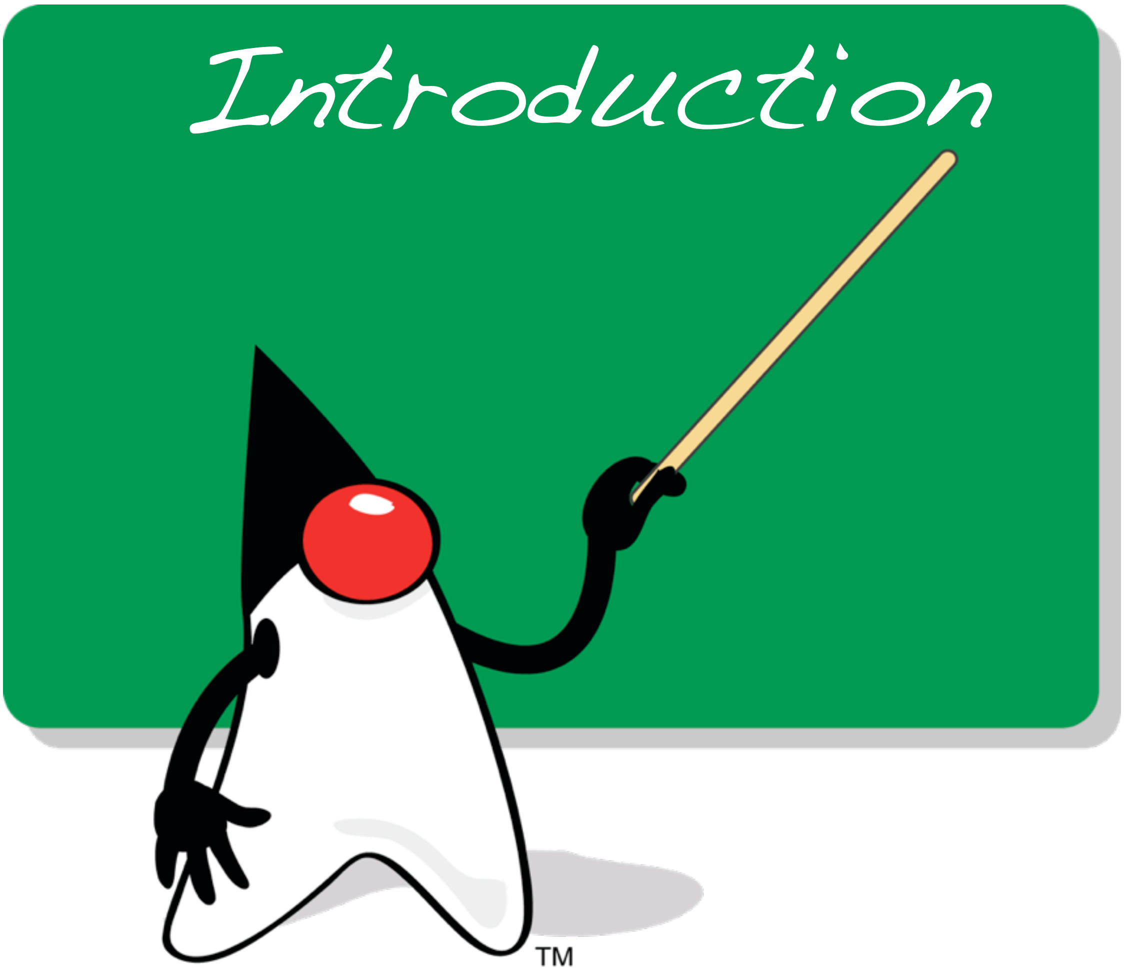
JavaServer Faces

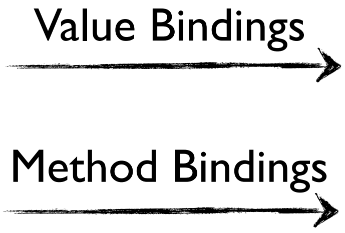
Introduction



JavaServer Faces

- Component based web framework
  - Build the UI from standard components
- Abstraction on the HTTP programming model
  - Event based, Swing like programming model
- Included in JEE
  - 1.2 -> JEE 5
  - 2.0 -> JEE 6

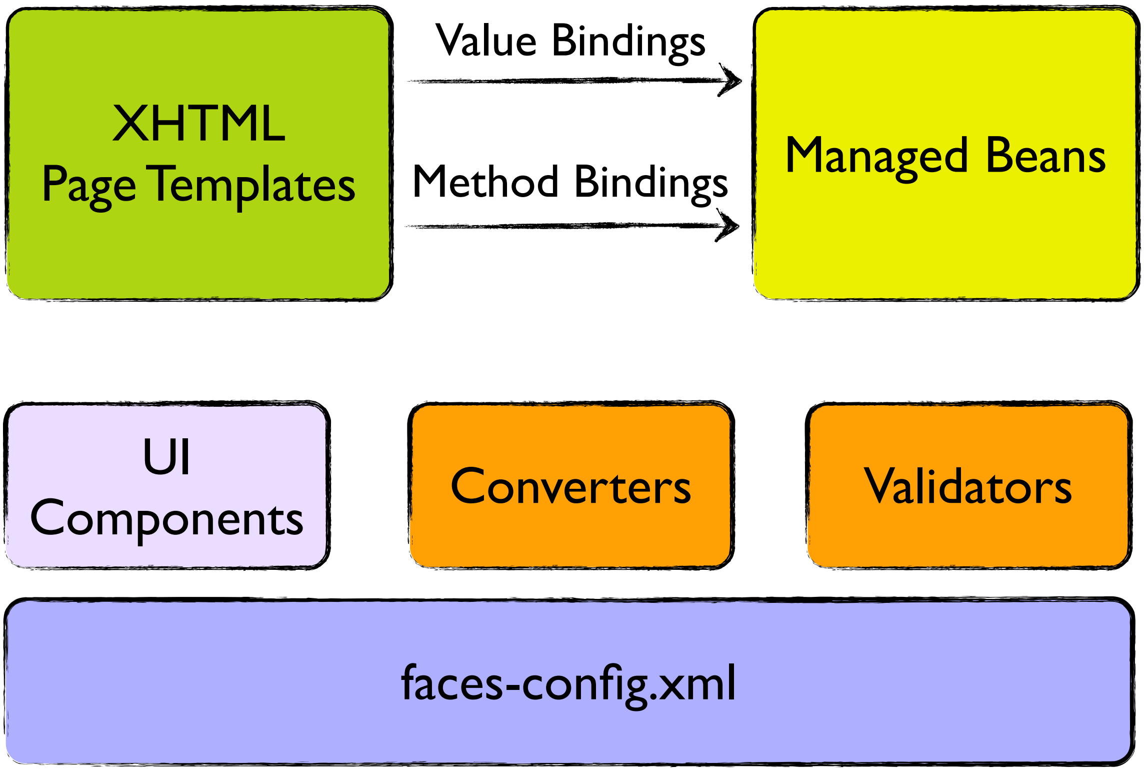
Architecture



Value Bindings

Method Bindings

Architecture



XHTML Page Templates

Value Bindings

Method Bindings

Managed Beans

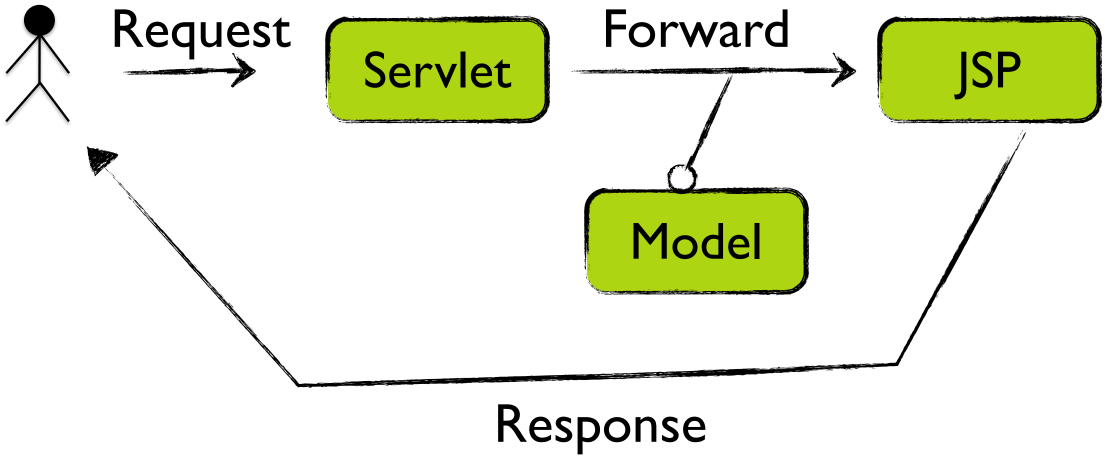
UI Components

Converters

Validators

faces-config.xml

Basic Model 2



Request

Servlet

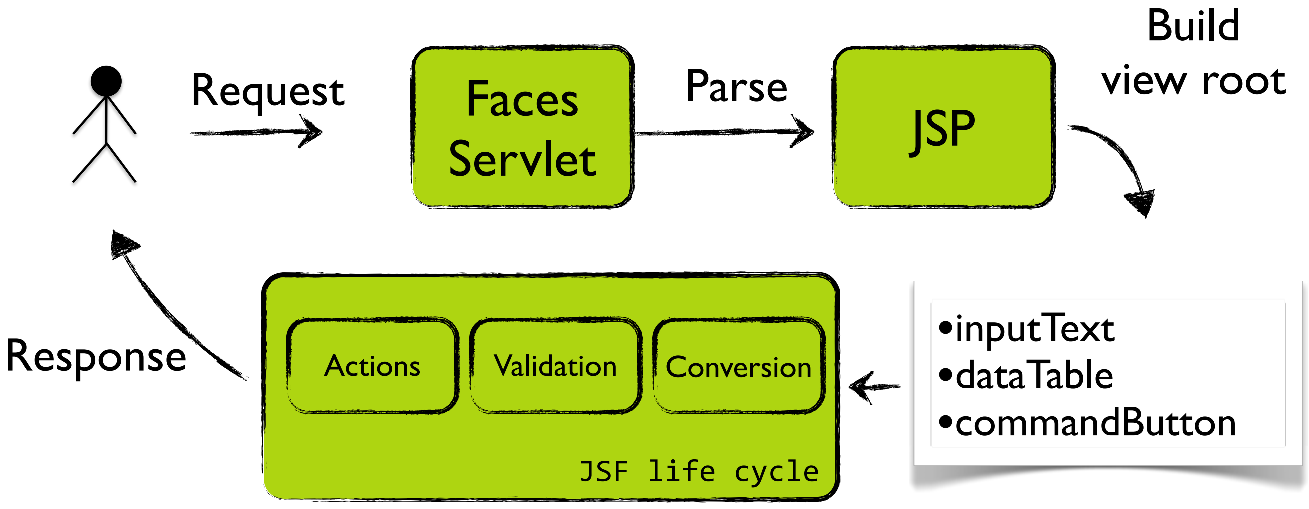
Forward

JSP

Model

Response

JSF



Request

Faces Servlet

Parse

JSP

Build view root

- inputText
- dataTable
- commandButton

Actions

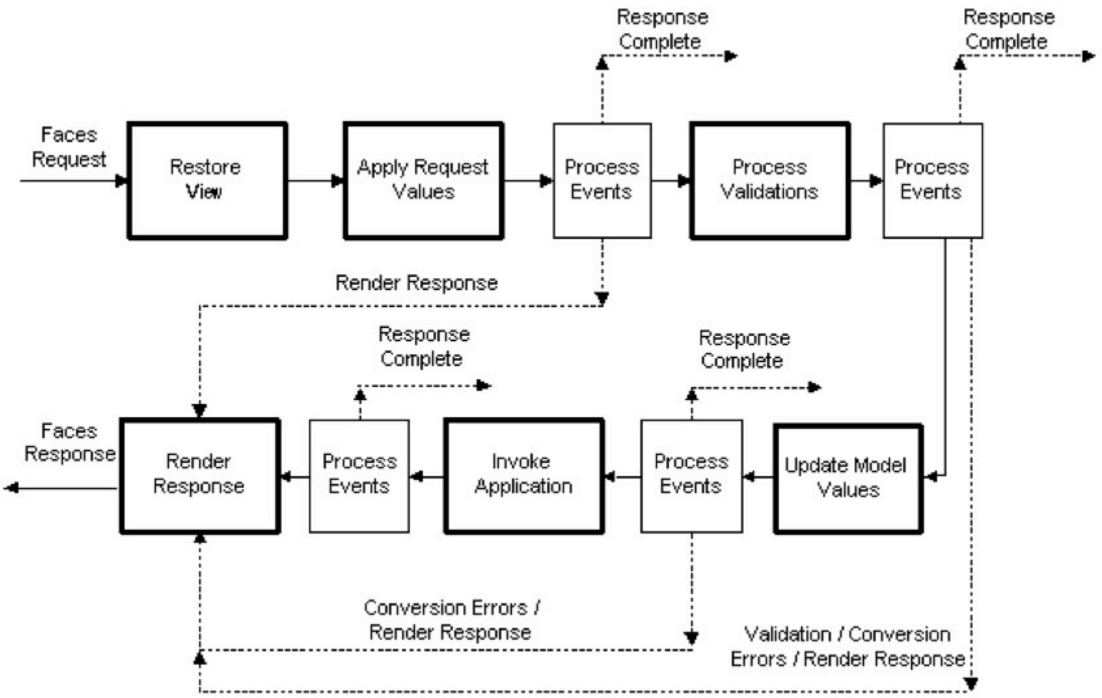
Validation

Conversion

JSF life cycle

Response

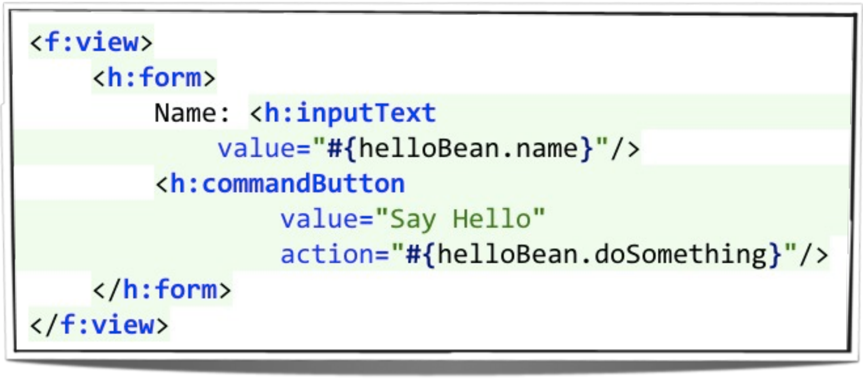
The life cycle



JSF views

- Definition of the view root
  - JSF will create an in-memory representation
- Build by JSF components using tags
- Combine with HTML for layout

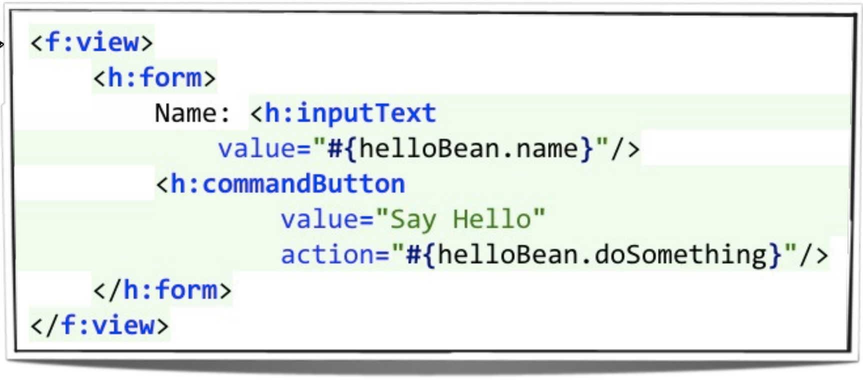
JSF view example



```
<f:view>
    <h:form>
        Name: <h:inputText
            value="#{helloBean.name}"/>
        <h:commandButton
                value="Say Hello"
                action="#{helloBean.doSomething}"/>
    </h:form>
</f:view>
```

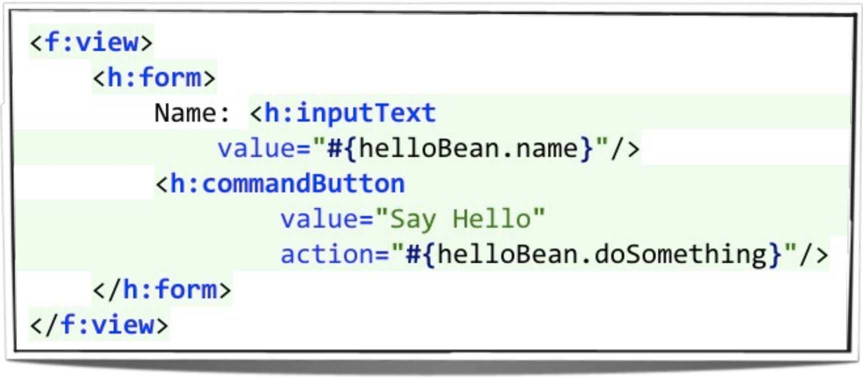
JSF view example

Start of view root



```
<f:view>
    <h:form>
        Name: <h:inputText
            value="#{helloBean.name}"/>
        <h:commandButton
                value="Say Hello"
                action="#{helloBean.doSomething}"/>
    </h:form>
</f:view>
```

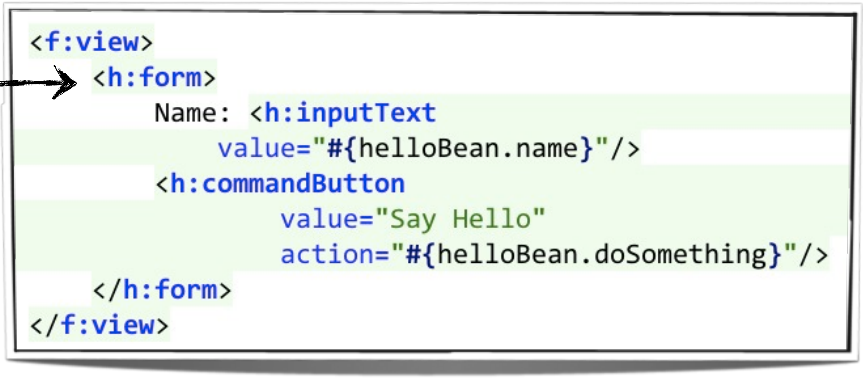
JSF view example



```
<f:view>
    <h:form>
        Name: <h:inputText
            value="#{helloBean.name}"/>
        <h:commandButton
                value="Say Hello"
                action="#{helloBean.doSomething}"/>
    </h:form>
</f:view>
```

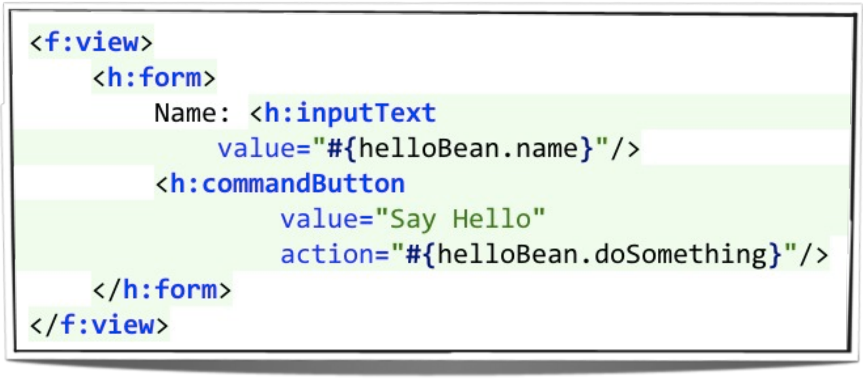
JSF view example

Necessary to send values back, and to execute commands



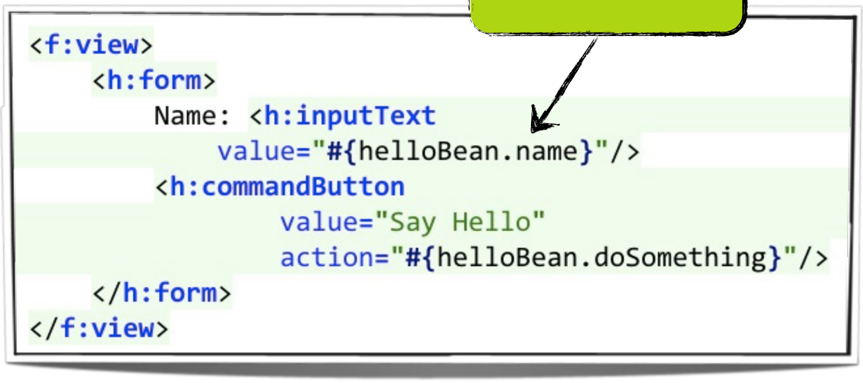
```
<f:view>
    <h:form>
        Name: <h:inputText
            value="#{helloBean.name}"/>
        <h:commandButton
                value="Say Hello"
                action="#{helloBean.doSomething}"/>
    </h:form>
</f:view>
```

JSF view example



```
<f:view>
    <h:form>
        Name: <h:inputText
            value="#{helloBean.name}"/>
        <h:commandButton
                value="Say Hello"
                action="#{helloBean.doSomething}"/>
    </h:form>
</f:view>
```

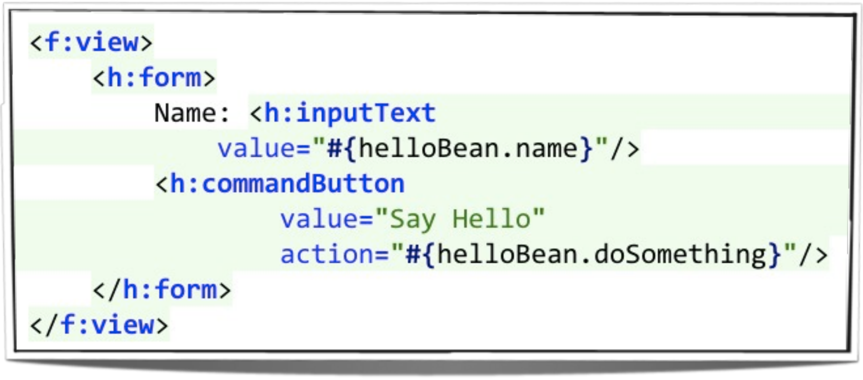
JSF view example



Value Binding

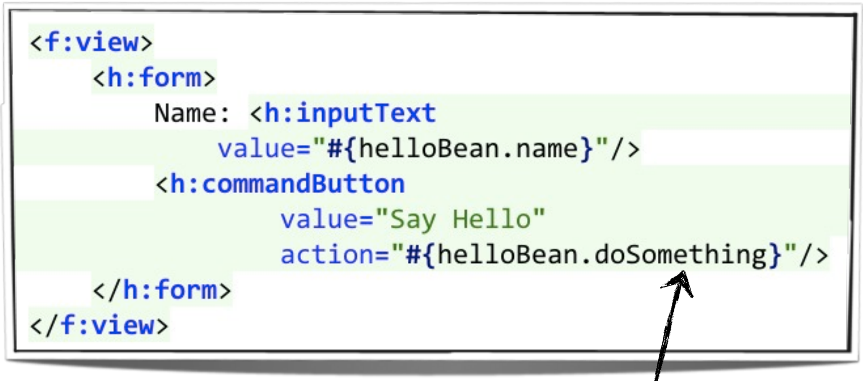
```
<f:view>
    <h:form>
        Name: <h:inputText
            value="#{helloBean.name}"/>
        <h:commandButton
                value="Say Hello"
                action="#{helloBean.doSomething}"/>
    </h:form>
</f:view>
```

JSF view example



```
<f:view>
    <h:form>
        Name: <h:inputText
            value="#{helloBean.name}"/>
        <h:commandButton
                value="Say Hello"
                action="#{helloBean.doSomething}"/>
    </h:form>
</f:view>
```

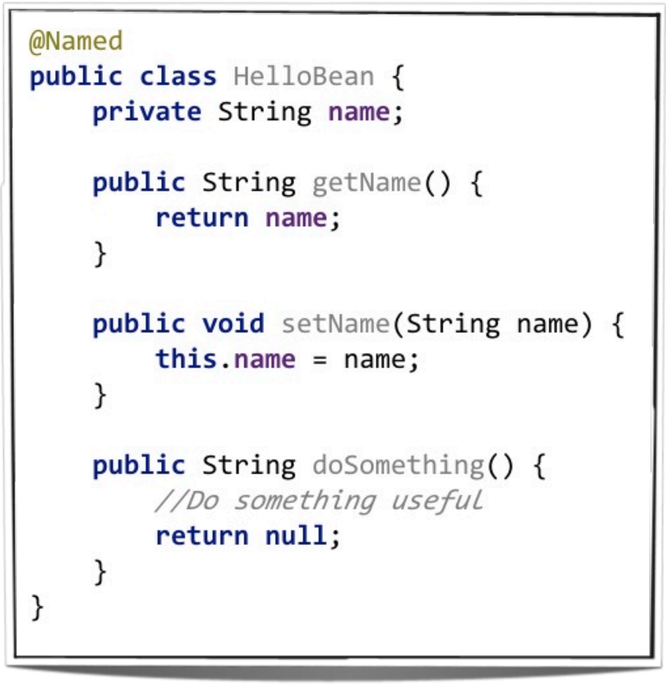
JSF view example



```
<f:view>
    <h:form>
        Name: <h:inputText
            value="#{helloBean.name}"/>
        <h:commandButton
                value="Say Hello"
                action="#{helloBean.doSomething}"/>
    </h:form>
</f:view>
```

Method Binding

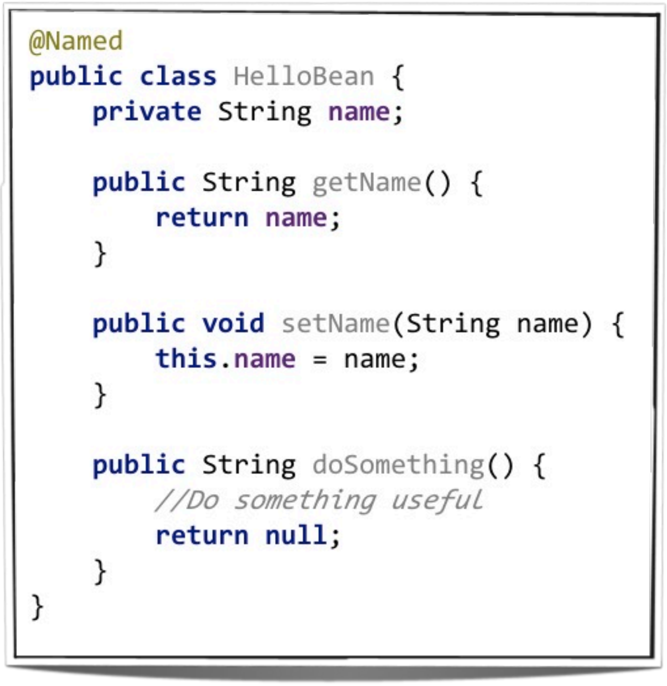
Managed Beans



```
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

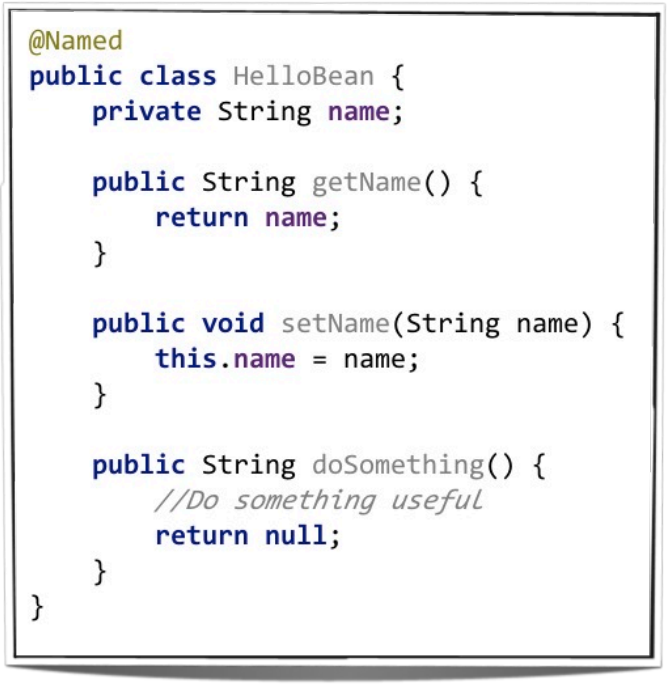
Managed Beans

Exposes bean to EL



```
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

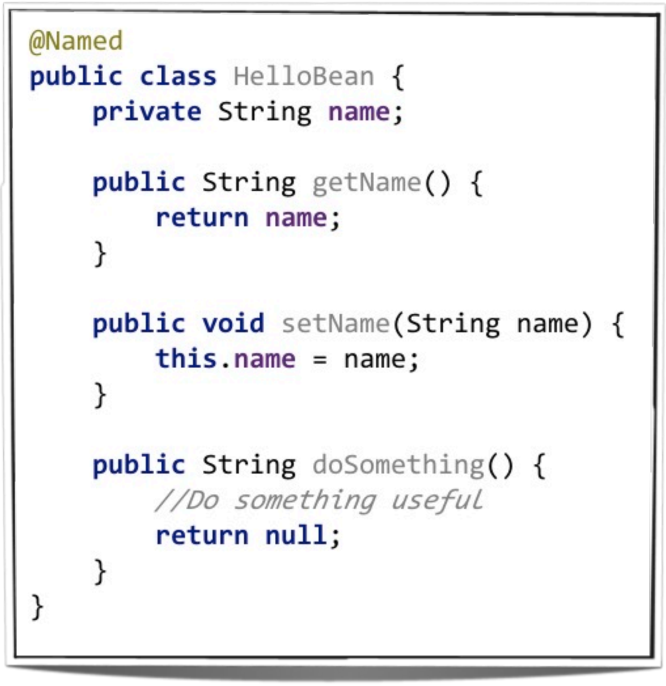
Managed Beans



```
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

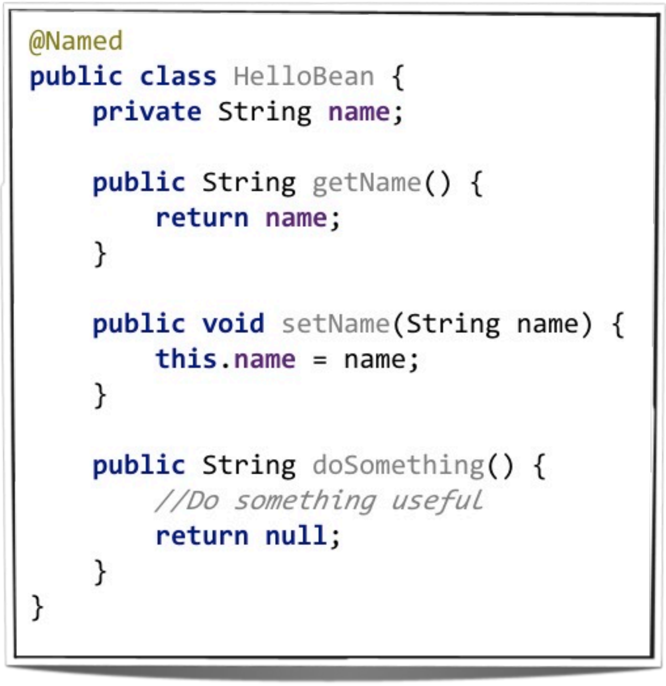
Managed Beans

No implements or extends



```
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

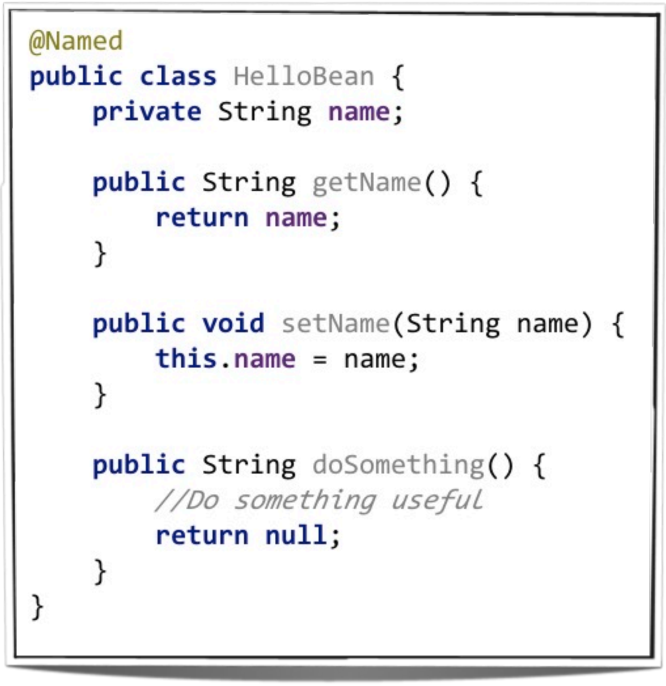
Managed Beans



```
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

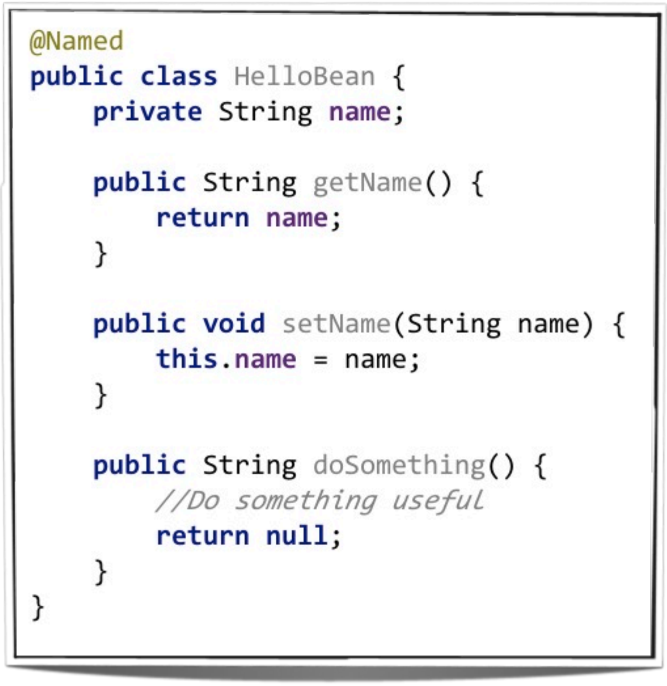
Managed Beans

Needed to render property



```
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

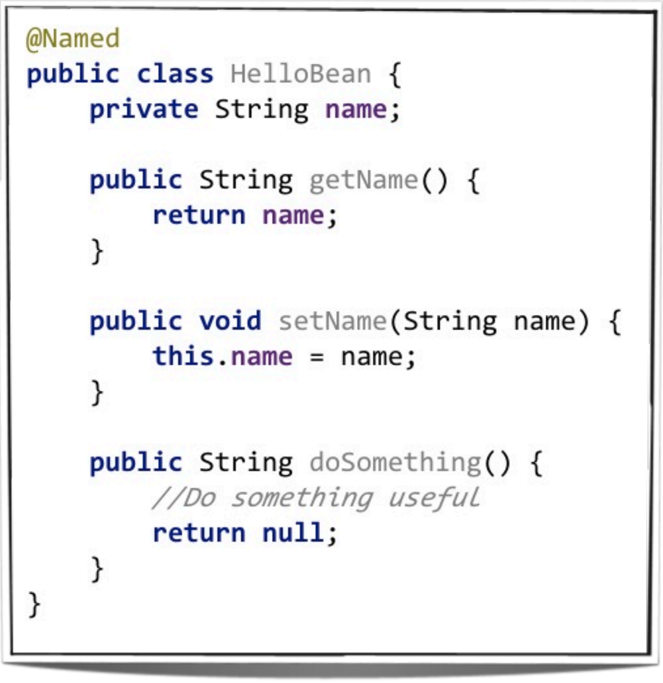
Managed Beans



```
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

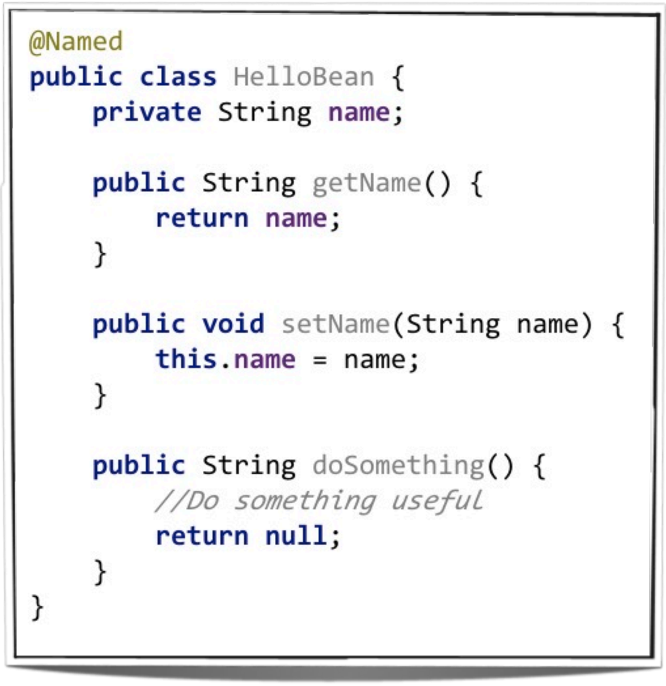
Managed Beans

Needed to submit property



```
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

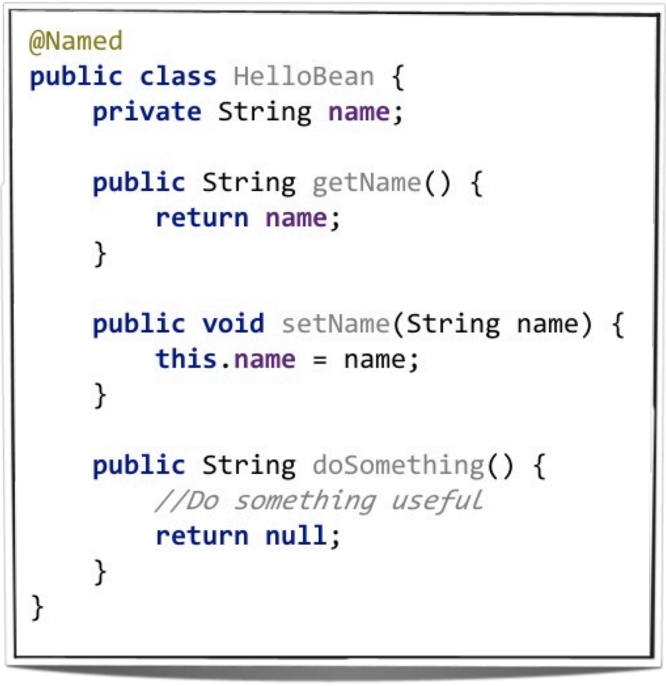
Managed Beans



```
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

Managed Beans

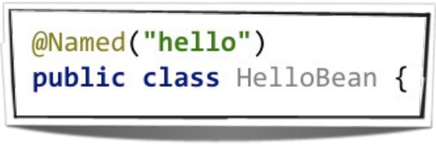
Action: must return String or Enum for navigation



```java
@Named
public class HelloBean {
    private String name;

    public String getName() {
        return name;
    }

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

    public String doSomething() {
        //Do something useful
        return null;
    }
}
```

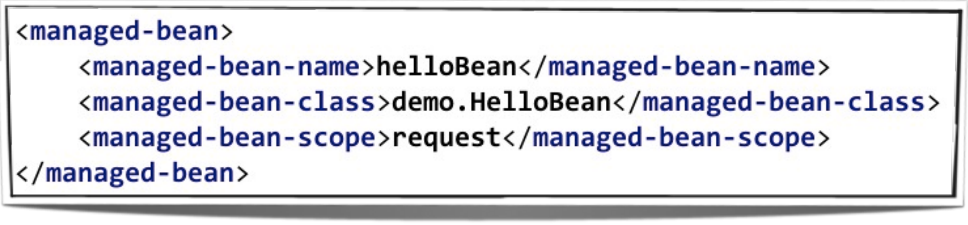
Managed Bean

- Use @Named in CDI (preferred!)



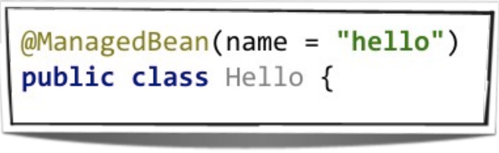
```java
@Named("hello")
public class HelloBean {
```

- Register in faces-config.xml



```xml
<managed-bean>
    <managed-bean-name>helloBean</managed-bean-name>
    <managed-bean-class>demo.HelloBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

- Use @ManagedBean



```java
@ManagedBean(name = "hello")
public class Hello {
```

Action methods

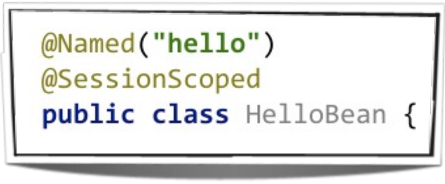
- Used with method binding
  - `action=”#{mybean.myAction}”`
  - `action=”#{mybean.myAction(‘arg1’)}”`
- Executed after the model is updated
  - See JSF life-cycle
- Must return String or Enum for navigation
  - Null to stay on same page

Value bindings

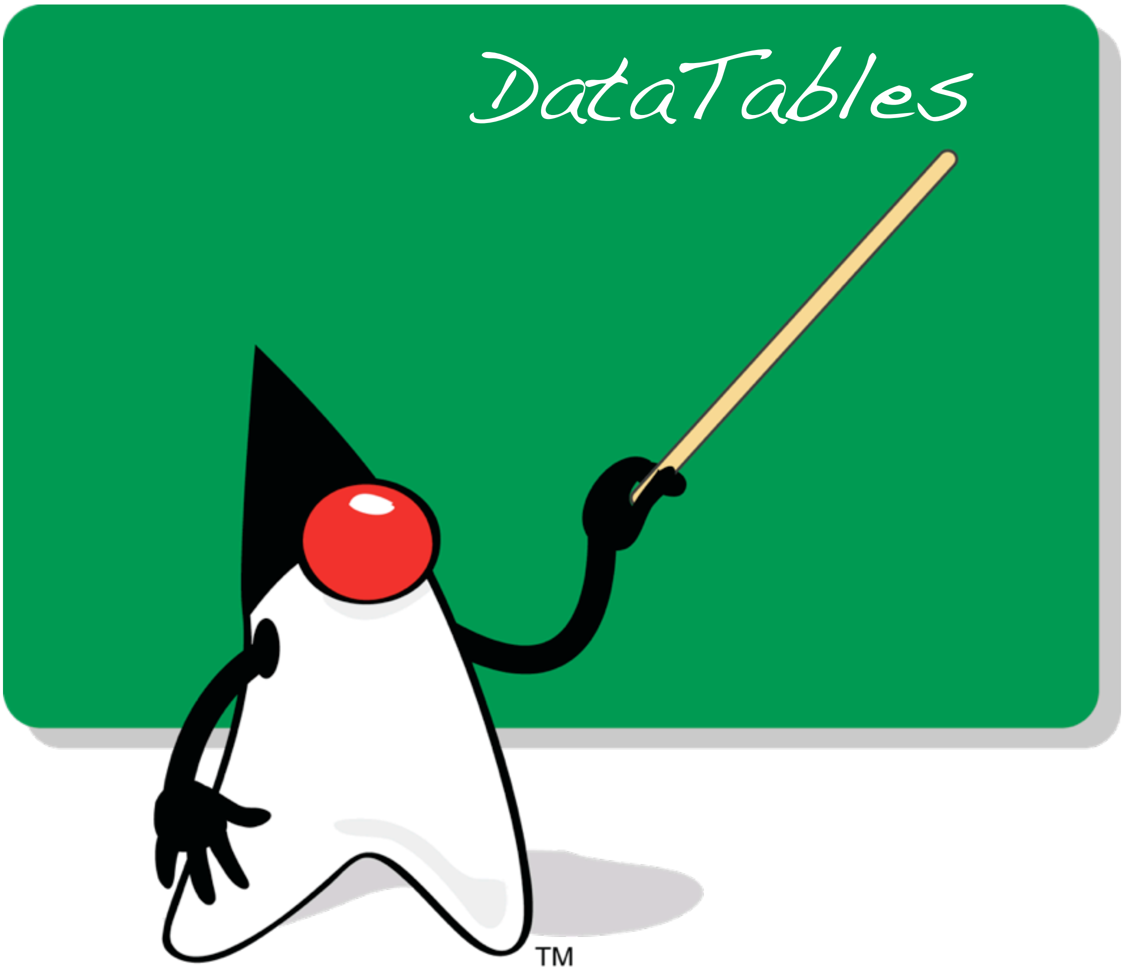
- Bind a field’s value to a property
  - use getter and setter
  - setter only required for input components
- `value=”#{mybean.myvalue}”`
  - binds to getValue() and setValue(value)
- `value=”#{mybean.myvalue()}”`
  - binds to myValue()
  - e.g. `#{bean.name.length()}`

States & Contexts

- CDI provides the following scopes
  - RequestScope
  - ConversationScope
  - SessionScope
  - ApplicationScope



```
@Named("hello")
@SessionScoped
public class HelloBean {
```



DataTables

h:dataTable

- JSF has a simple dataTable component
  - renders to a HTML table
  - renders from different kind of data sets
- No out-of-the-box support for advanced features
  - no paging, sorting, filtering etc.

h:dataTable types

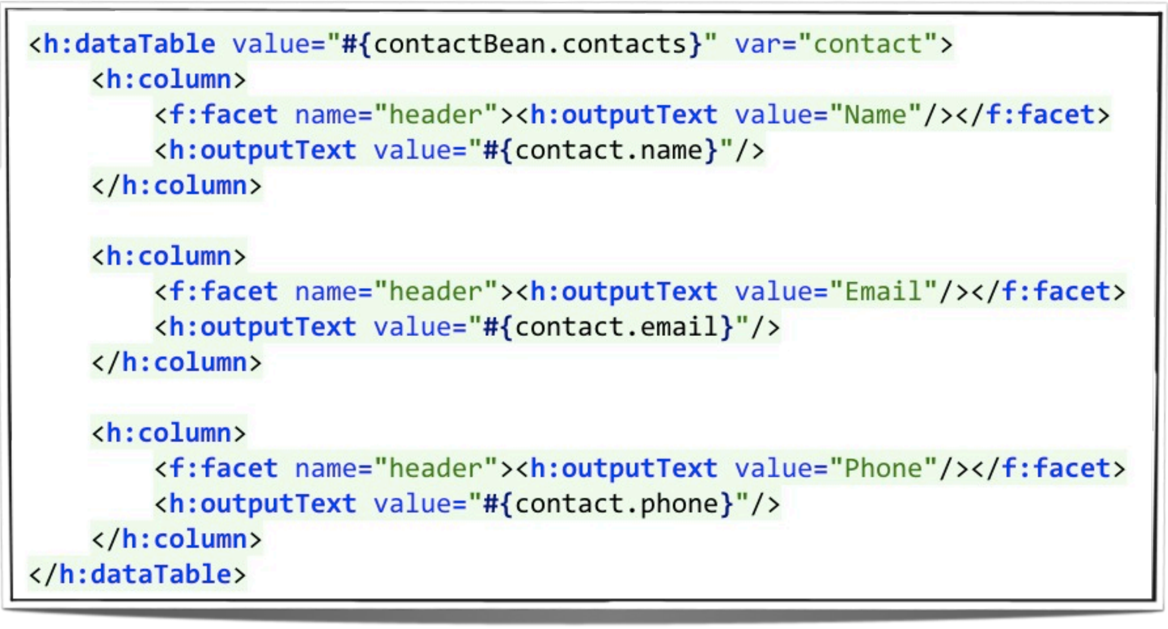
- Array
- java.util.List
- java.sql.ResultSet
- javax.servlet.jsp.jstl.sql.Result
- javax.faces.model.DataModel
  - wraps a List, adds "selectedRow"
  - used most of the time

h:dataTable types

- Array
- java.util.List
- ~~java.sql.ResultSet~~
- ~~javax.servlet.jsp.jstl.sql.Result~~
- javax.faces.model.DataModel
  - wraps a List, adds “selectedRow”
  - used most of the time

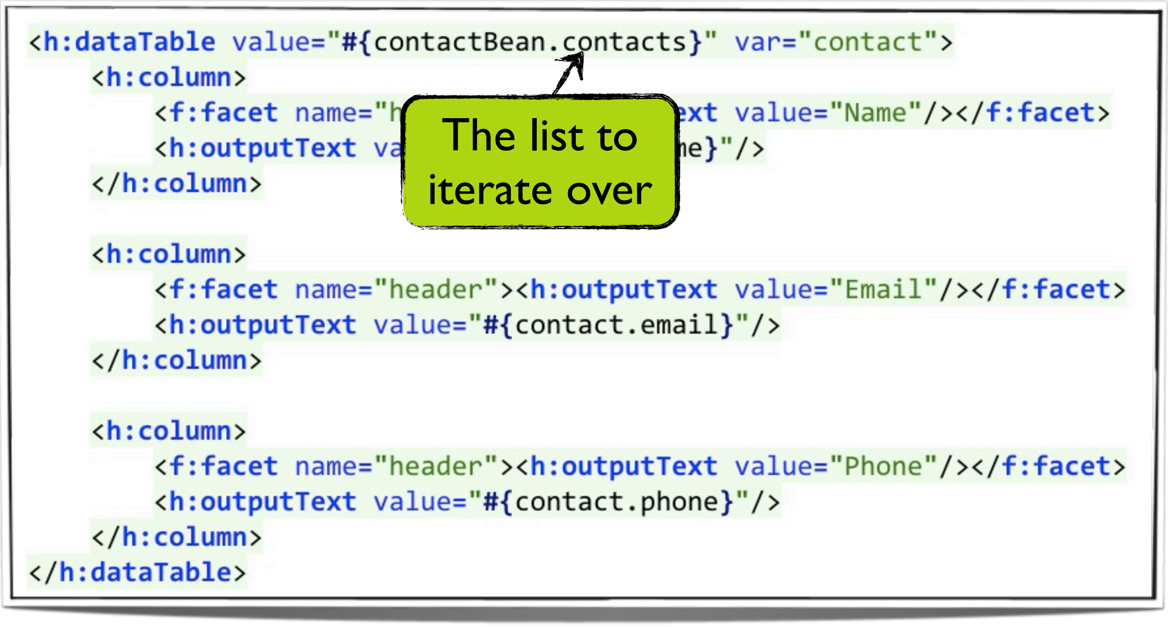
Don’t use JDBC in the view....

h:dataTable example



```
<h:dataTable value="#{contactBean.contacts}" var="contact">
    <h:column>
        <f:facet name="header"><h:outputText value="Name"/></f:facet>
        <h:outputText value="#{contact.name}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

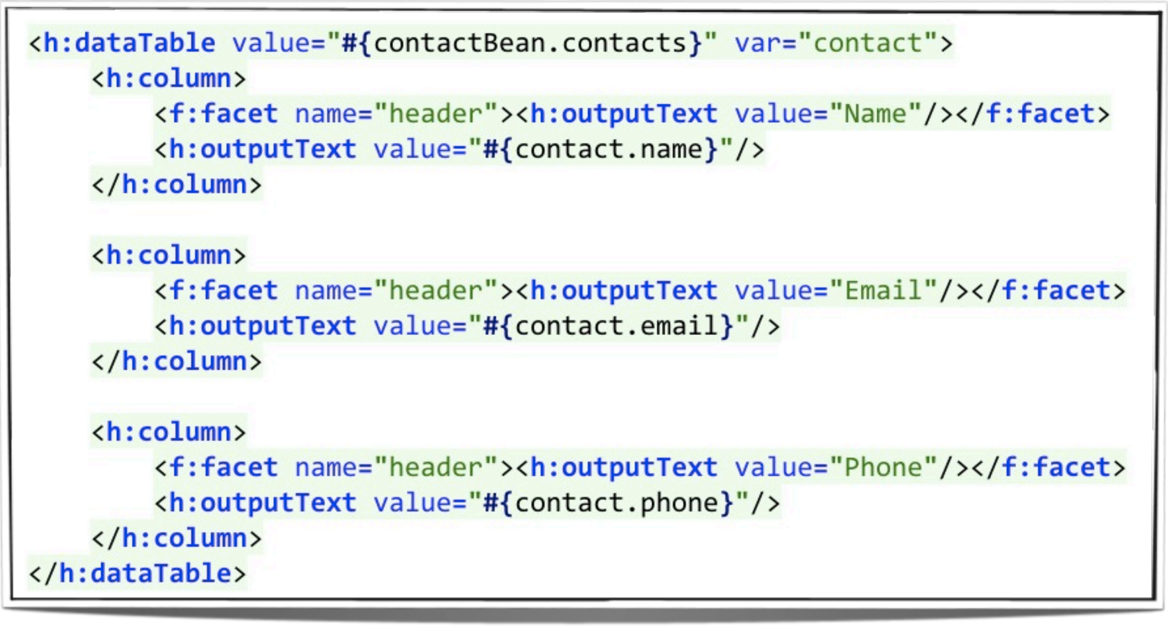
h:dataTable example



```
<h:dataTable value="#{contactBean.contacts}" var="contact">
    <h:column>
        <f:facet name="h                  ext value="Name"/></f:facet>
        <h:outputText va                  e}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

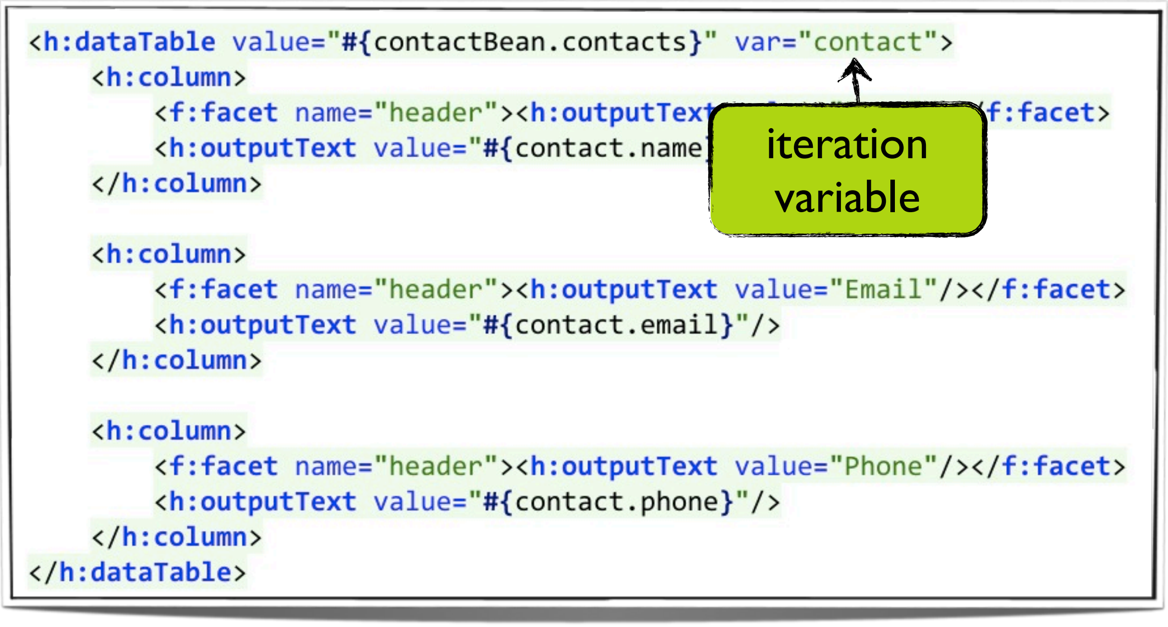
The list to iterate over

h:dataTable example



```
<h:dataTable value="#{contactBean.contacts}" var="contact">
    <h:column>
        <f:facet name="header"><h:outputText value="Name"/></f:facet>
        <h:outputText value="#{contact.name}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

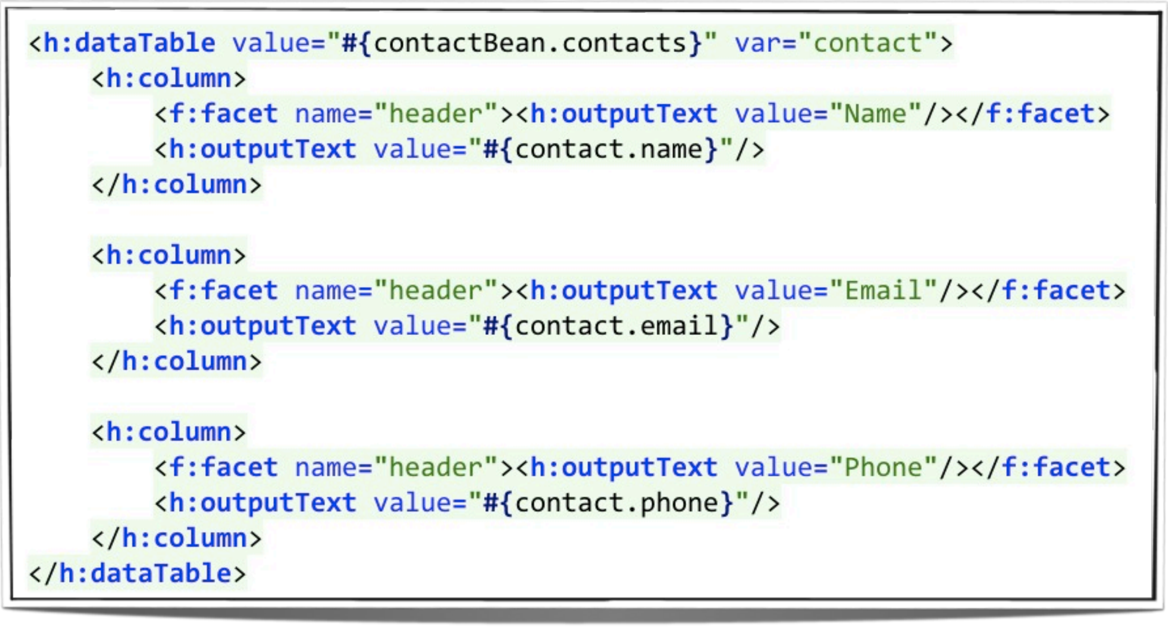
h:dataTable example



```
<h:dataTable value="#{contactBean.contacts}" var="contact">
    <h:column>
        <f:facet name="header"><h:outputText                 f:facet>
        <h:outputText value="#{contact.name}
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

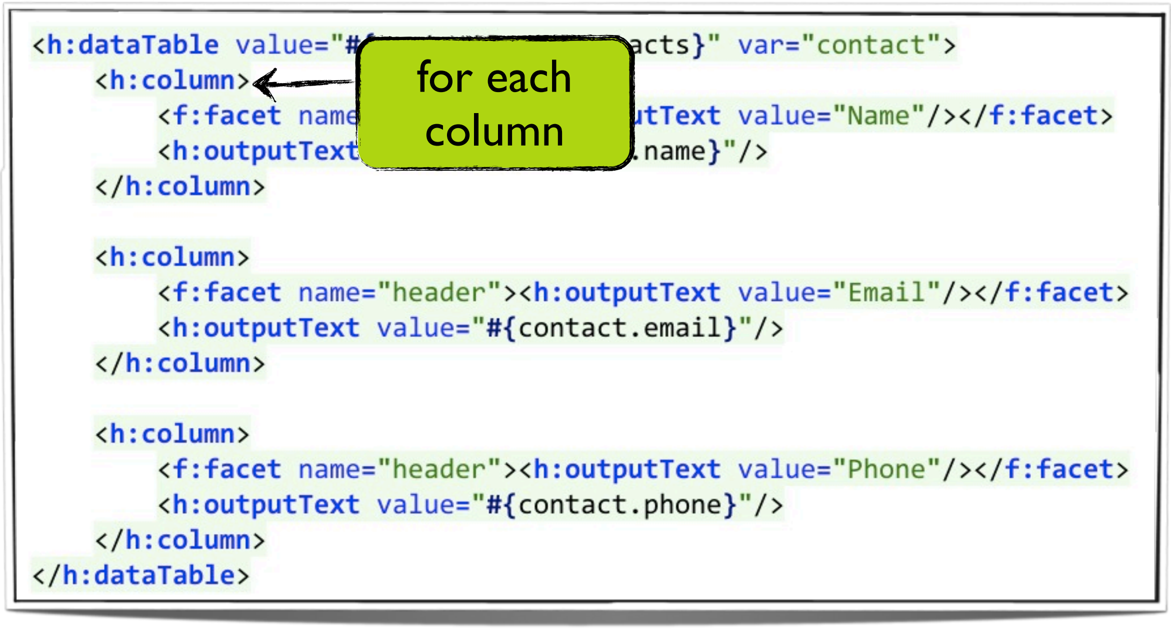
iteration variable

h:dataTable example



```
<h:dataTable value="#{contactBean.contacts}" var="contact">
    <h:column>
        <f:facet name="header"><h:outputText value="Name"/></f:facet>
        <h:outputText value="#{contact.name}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

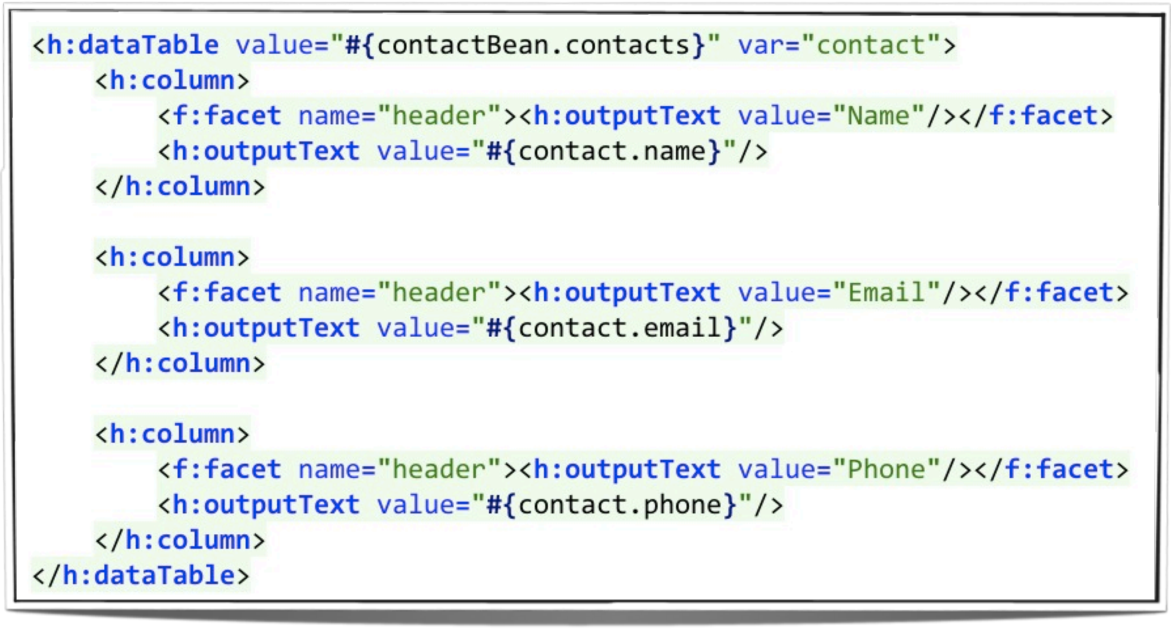
h:dataTable example



```
<h:dataTable value="#                acts}" var="contact">
    <h:column>
        <f:facet name                utText value="Name"/></f:facet>
        <h:outputText                .name}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

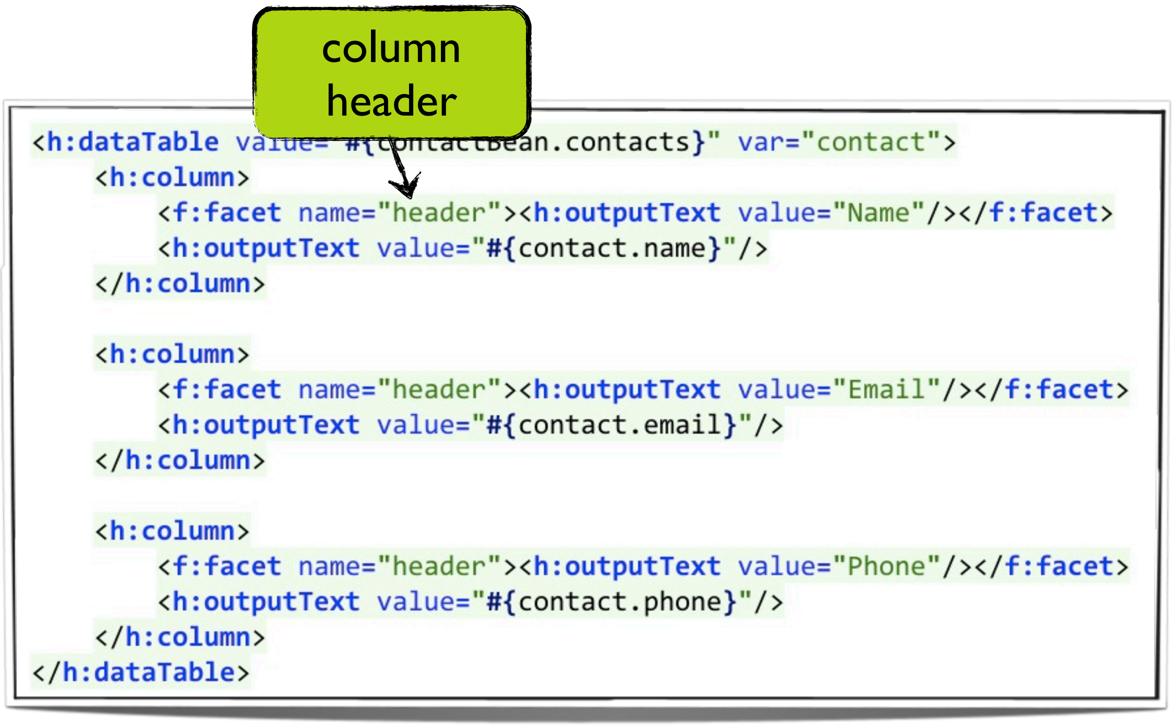
for each column

h:dataTable example



```
<h:dataTable value="#{contactBean.contacts}" var="contact">
    <h:column>
        <f:facet name="header"><h:outputText value="Name"/></f:facet>
        <h:outputText value="#{contact.name}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

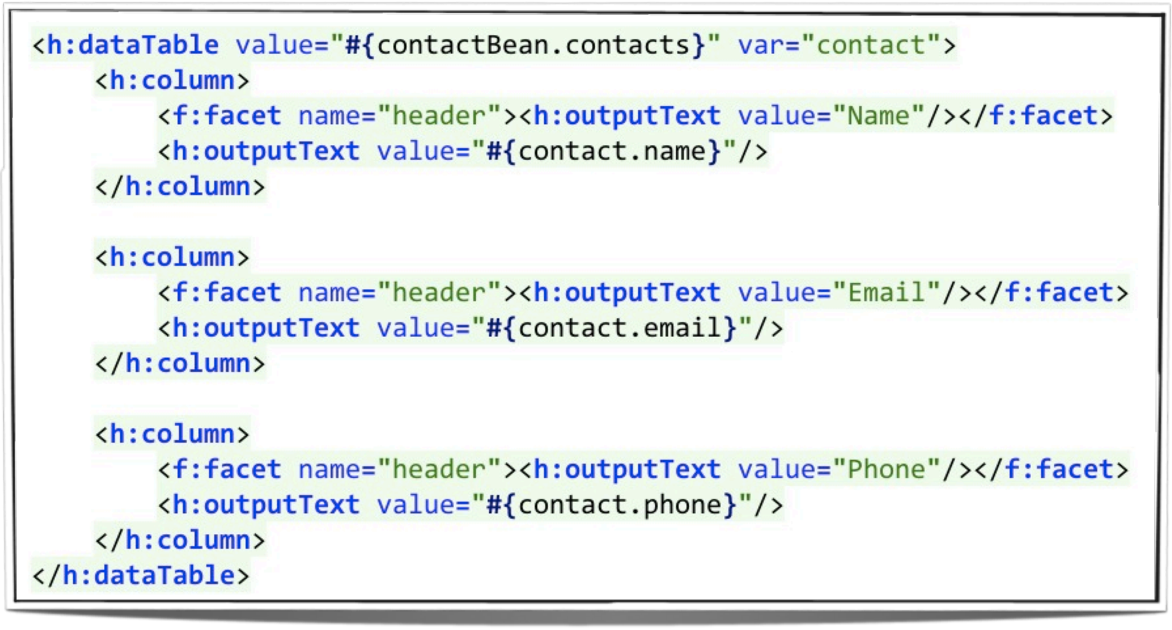
h:dataTable example



column header

```
<h:dataTable value="#{contactBean.contacts}" var="contact">
    <h:column>
        <f:facet name="header"><h:outputText value="Name"/></f:facet>
        <h:outputText value="#{contact.name}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

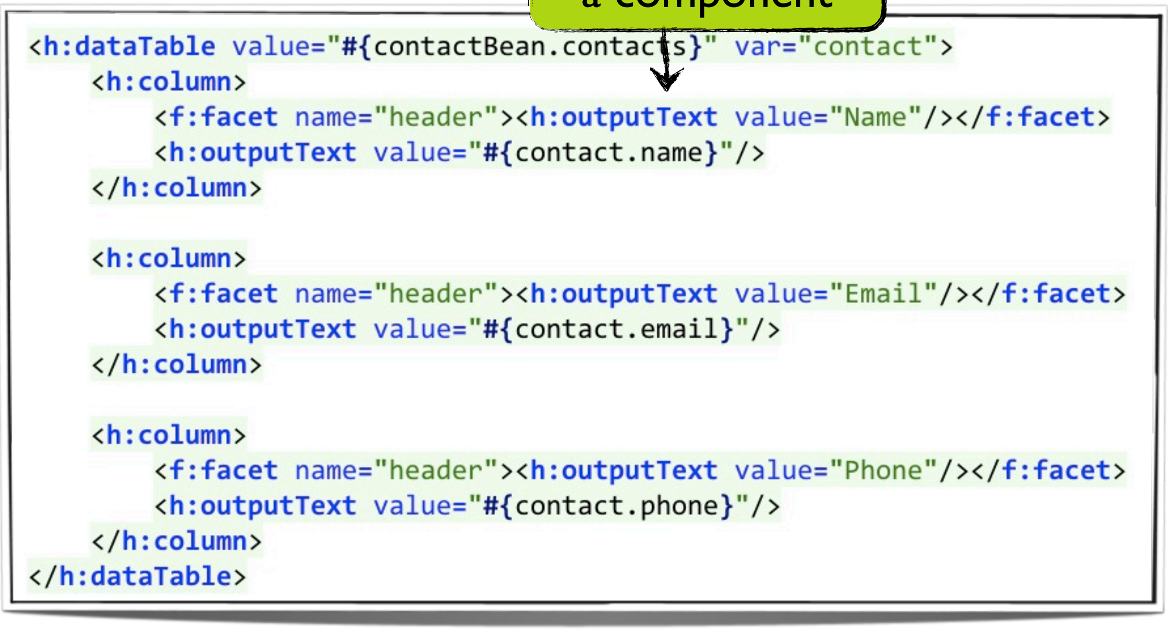
h:dataTable example



```
<h:dataTable value="#{contactBean.contacts}" var="contact">
    <h:column>
        <f:facet name="header"><h:outputText value="Name"/></f:facet>
        <h:outputText value="#{contact.name}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

h:dataTable example

header must be a component

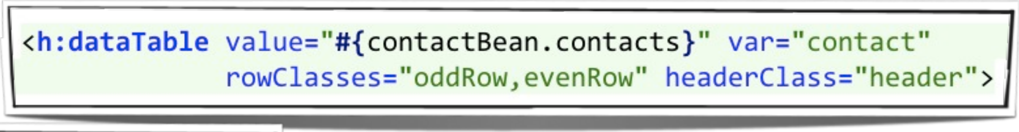


```
<h:dataTable value="#{contactBean.contacts}" var="contact">
    <h:column>
        <f:facet name="header"><h:outputText value="Name"/></f:facet>
        <h:outputText value="#{contact.name}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Email"/></f:facet>
        <h:outputText value="#{contact.email}"/>
    </h:column>

    <h:column>
        <f:facet name="header"><h:outputText value="Phone"/></f:facet>
        <h:outputText value="#{contact.phone}"/>
    </h:column>
</h:dataTable>
```

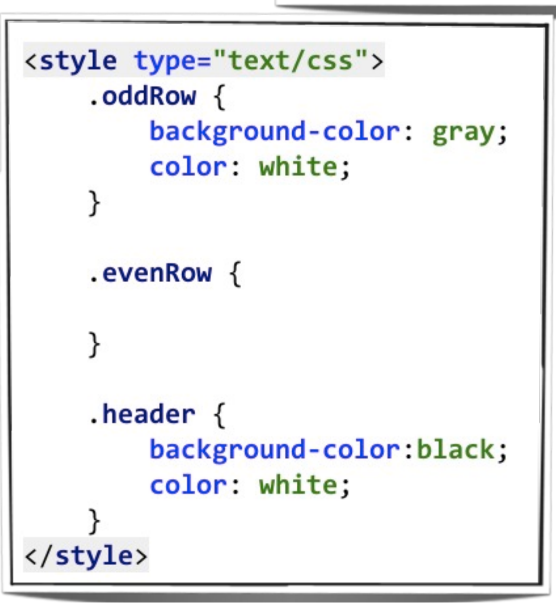
Styling tables

- Specify style at different levels
  - whole table (styleClass)
  - headers and footers (headerClass, footerclass)
  - columns (columnClasses)
  - rows (rowClasses)

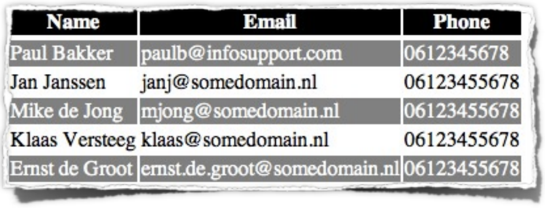
Styling tables



```
<h:dataTable value="#{contactBean.contacts}" var="contact"
             rowClasses="oddRow,evenRow" headerClass="header">
```



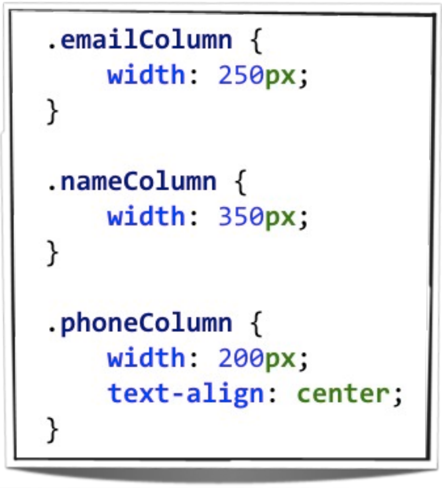
```
<style type="text/css">
    .oddRow {
        background-color: gray;
        color: white;
    }

    .evenRow {

    }

    .header {
        background-color:black;
        color: white;
    }
</style>
```



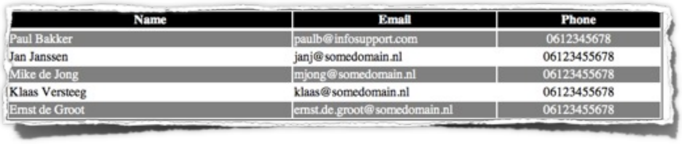
Name	Email	Phone
Paul Bakker	paulb@infosupport.com	0612345678
Jan Janssen	janj@somedomain.nl	06123455678
Mike de Jong	mjong@somedomain.nl	06123455678
Klaas Versteeg	klaas@somedomain.nl	06123455678
Ernst de Groot	ernst.de.groot@somedomain.nl	06123455678

Column width

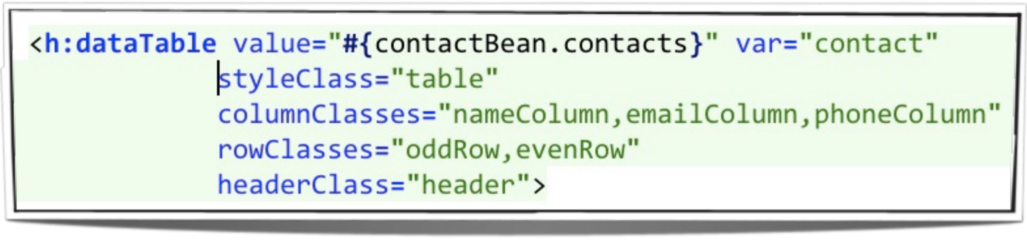
- ❑ Specify style for each column



```
.emailColumn {
    width: 250px;
}

.nameColumn {
    width: 350px;
}

.phoneColumn {
    width: 200px;
    text-align: center;
}
```



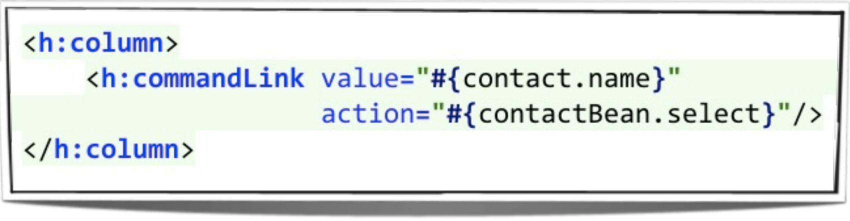
Name	Email	Phone
Paul Bakker	paulb@infosupport.com	0612345678
Jan Janssen	janj@somedomain.nl	06123455678
Mike de Jong	mjong@somedomain.nl	06123455678
Klaas Versteeg	klaas@somedomain.nl	06123455678
Ernst de Groot	ernst.de.groot@somedomain.nl	06123455678



```
<h:dataTable value="#{contactBean.contacts}" var="contact"
             styleClass="table"
             columnClasses="nameColumn,emailColumn,phoneColumn"
             rowClasses="oddRow,evenRow"
             headerClass="header">
```

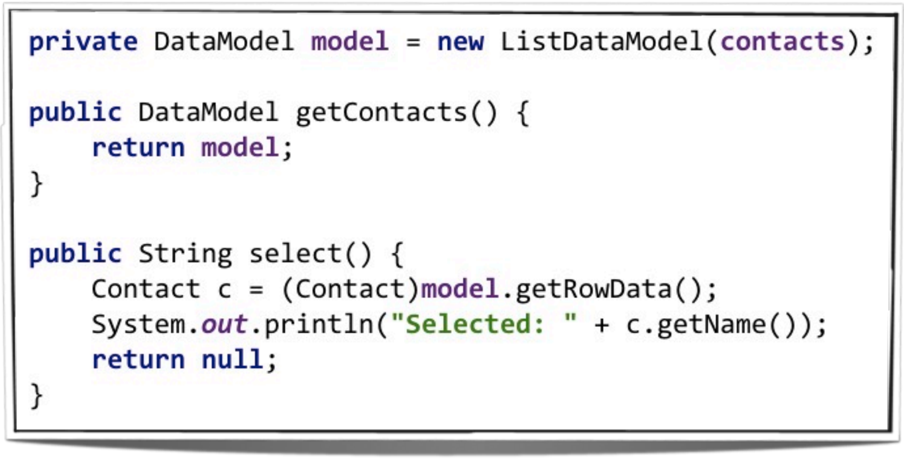
Selecting rows

- Wrap the data in a DataModel
  - Provides access to the selected row
- Handle selection in action method



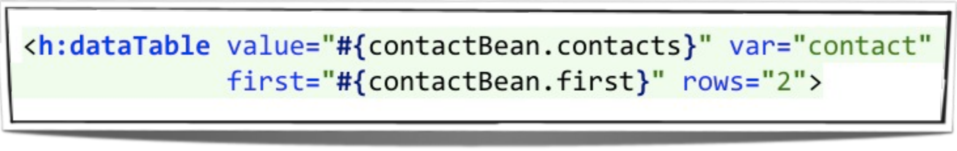
```
<h:column>
    <h:commandLink value="#{contact.name}"
                   action="#{contactBean.select}"/>
</h:column>
```

Selecting rows

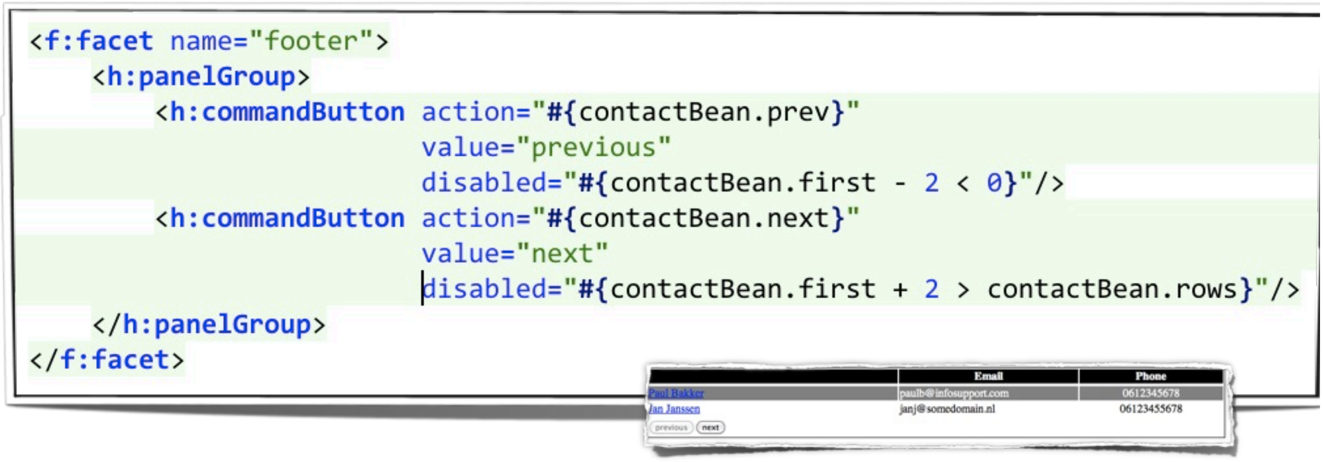


```
private DataModel model = new ListDataModel(contacts);

public DataModel getContacts() {
    return model;
}

public String select() {
    Contact c = (Contact)model.getRowData();
    System.out.println("Selected: " + c.getName());
    return null;
}
```

Paging



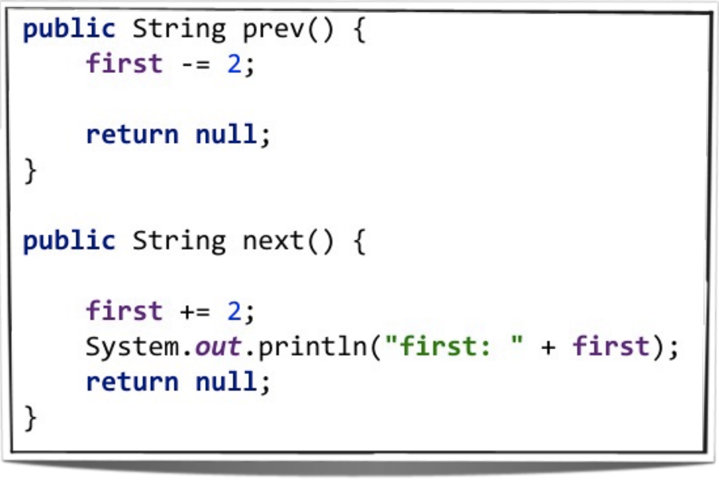
```
<h:dataTable value="#{contactBean.contacts}" var="contact"
             first="#{contactBean.first}" rows="2">
```



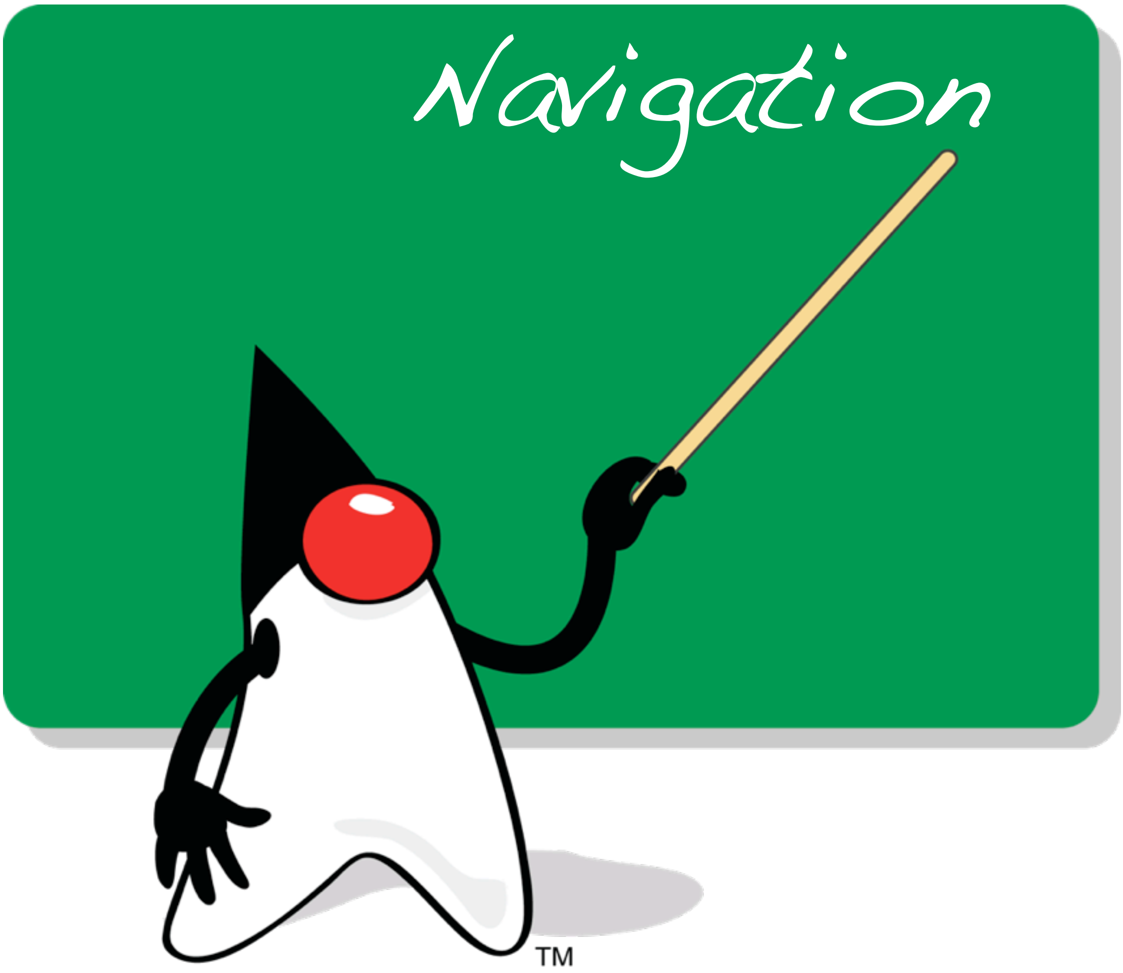
```
<f:facet name="footer">
    <h:panelGroup>
        <h:commandButton action="#{contactBean.prev}"
                         value="previous"
                         disabled="#{contactBean.first - 2 < 0}"/>
        <h:commandButton action="#{contactBean.next}"
                         value="next"
                         disabled="#{contactBean.first + 2 > contactBean.rows}"/>
    </h:panelGroup>
</f:facet>
```

	Email	Phone
Paul Bakker	paulb@infosupport.com	0612345678
Jan Janssen	janj@somedomain.nl	06123455678

Paging



```
public String prev() {
    first -= 2;

    return null;
}

public String next() {

    first += 2;
    System.out.println("first: " + first);
    return null;
}
```



Navigation

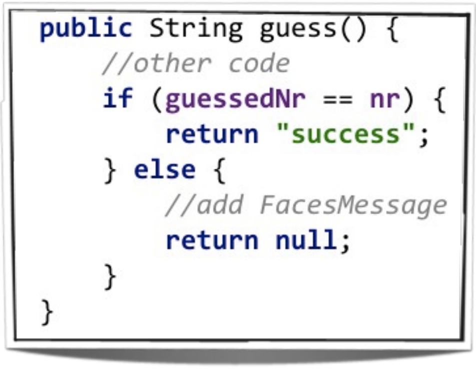
- Action methods trigger navigation
- Implicit navigation
  - use the page name to navigate
- Explicit navigation
  - faces-config.xml navigation rules

Explicit Navigation

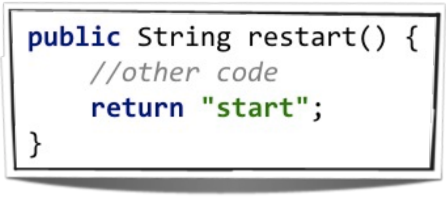
- View-to-view navigation as the result of actions
- Configured in faces-config.xml
- Server side forward by default
  - Request scoped state still available on new page
  - Not bookmarkable

Navigation example

- Guess a number example
  - when guessed right: navigate to success.jsp
  - Stay on the same page otherwise

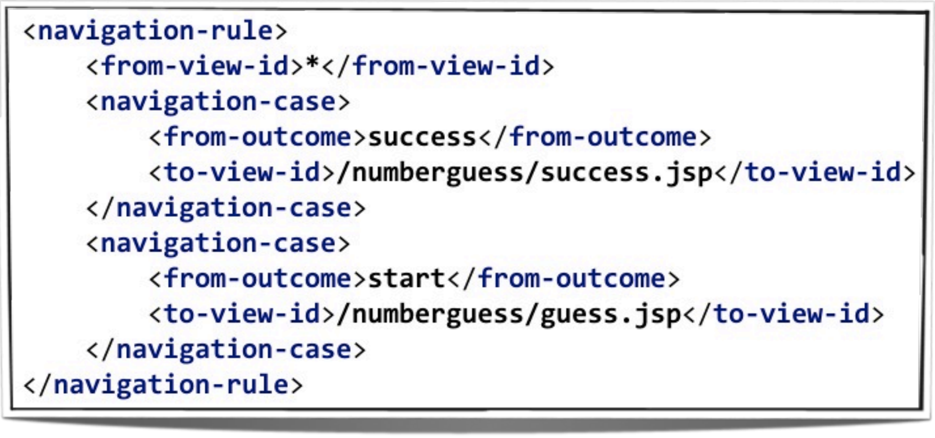


```
public String guess() {
    //other code
    if (guessedNr == nr) {
        return "success";
    } else {
        //add FacesMessage
        return null;
    }
}
```



```
public String restart() {
    //other code
    return "start";
}
```

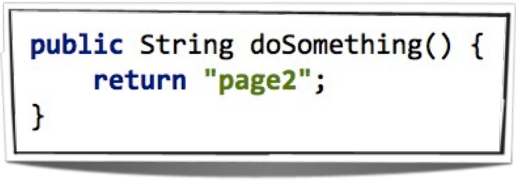
Navigation rules



```
<navigation-rule>
    <from-view-id>*</from-view-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/numberguess/success.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>start</from-outcome>
        <to-view-id>/numberguess/guess.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

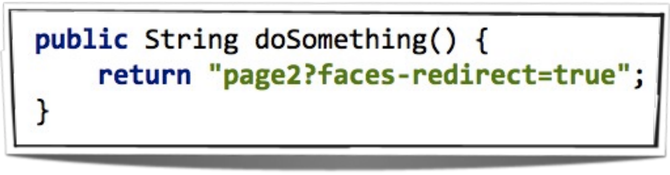
Implicit navigation

- □ Return the view-id directly



```
public String doSomething() {
    return "page2";
}
```

- □ Optionally use a redirect

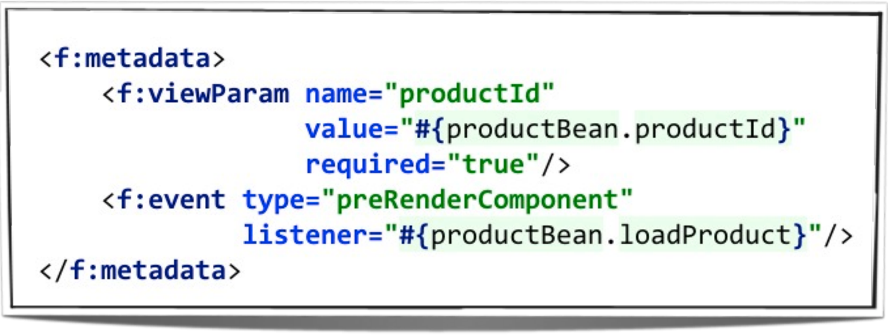


```
public String doSomething() {
    return "page2?faces-redirect=true";
}
```

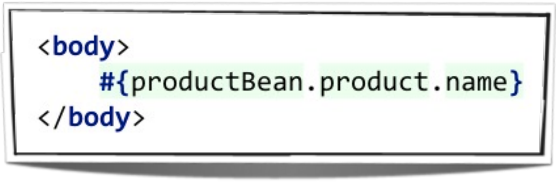
Deep linking

- Linking to a page with specific state
  - e.g. `productdetails.jsp?pid=10` shows the detail page of product with pid=10
  - very common web functionality
  - enables deep bookmarking

Pulling data before rendering

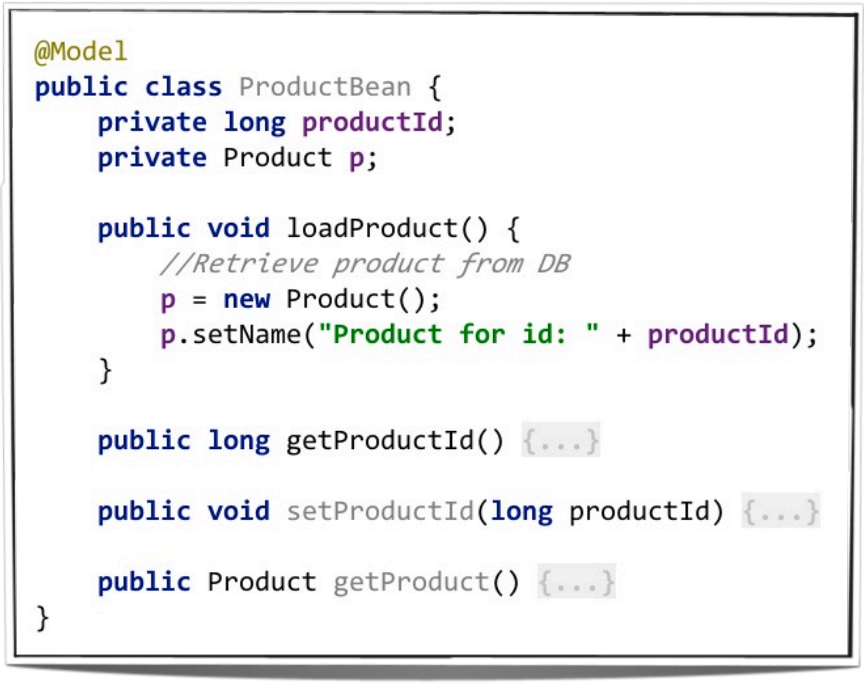


```
<f:metadata>
    <f:viewParam name="productId"
                 value="#{productBean.productId}"
                 required="true"/>
    <f:event type="preRenderComponent"
             listener="#{productBean.loadProduct}"/>
</f:metadata>
```



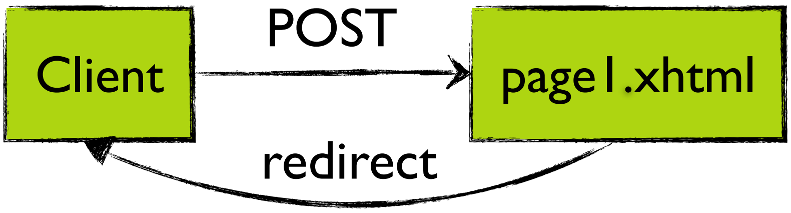
```
<body>
    #{productBean.product.name}
</body>
```

Pre-loading data

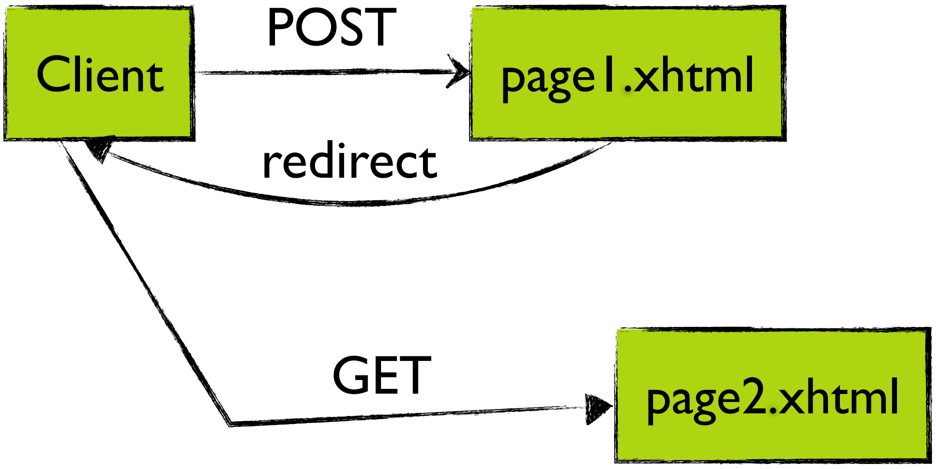


```
@Model
public class ProductBean {
    private long productId;
    private Product p;

    public void loadProduct() {
        //Retrieve product from DB
        p = new Product();
        p.setName("Product for id: " + productId);
    }

    public long getProductId() {...}

    public void setProductId(long productId) {...}

    public Product getProduct() {...}
}
```

Post-Redirect-Get

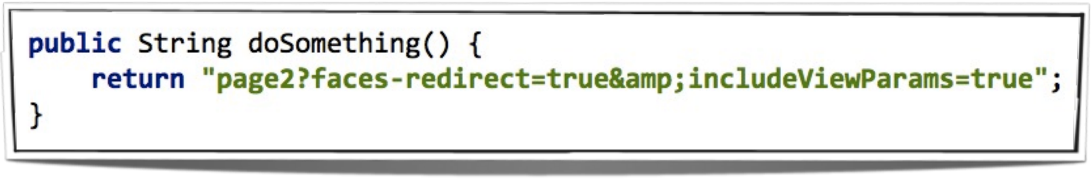


Post-Redirect-Get



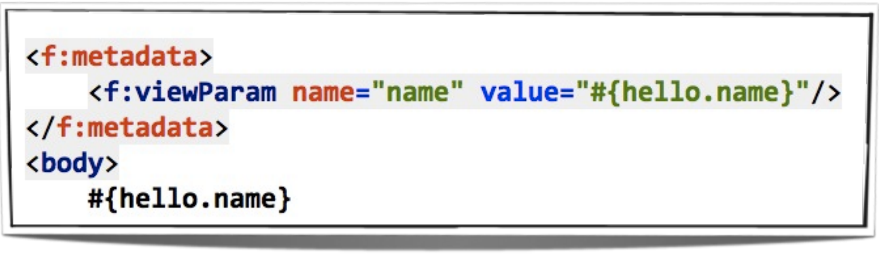
Post-Redirect-Get

- State should be passed in the url of the second page
  - e.g. `page2.xhtml?name=Paul`

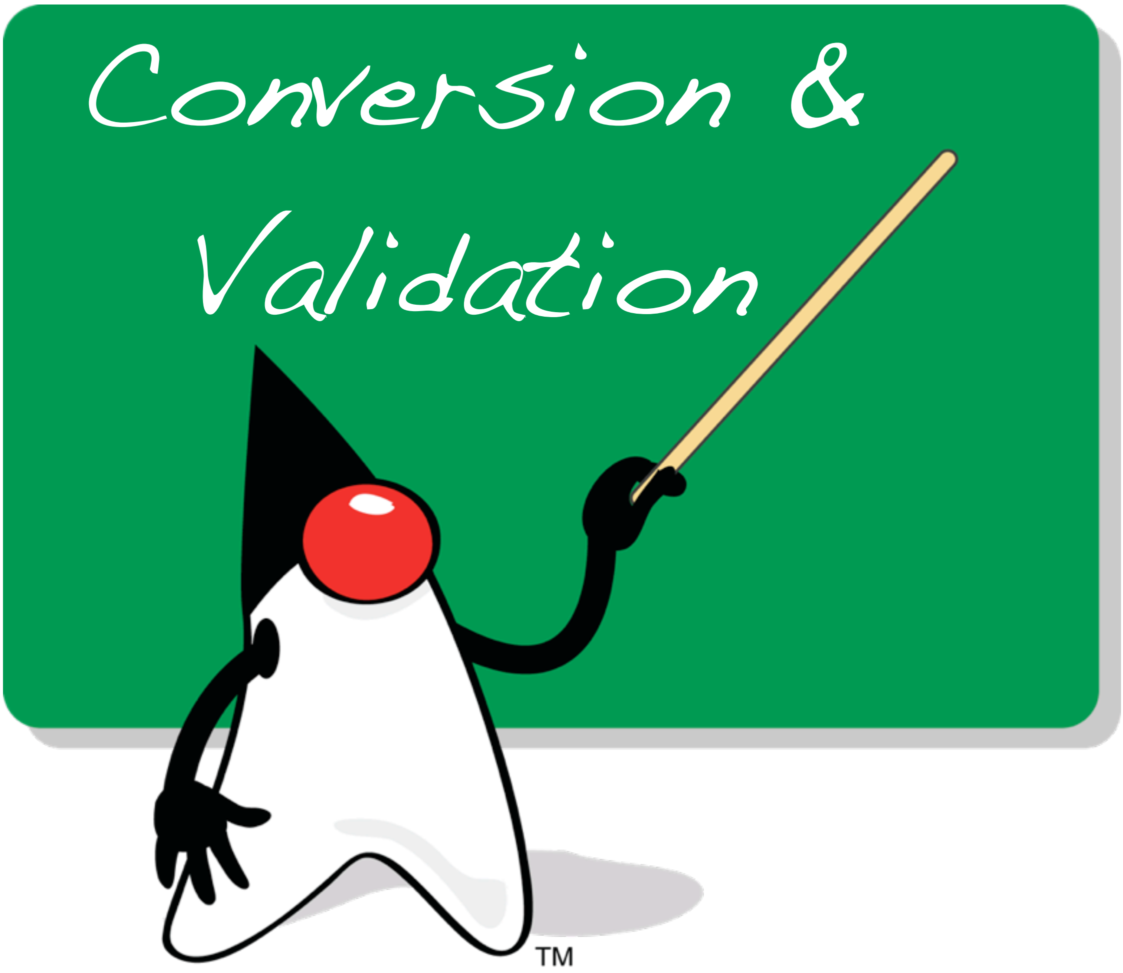


```
public String doSomething() {
    return "page2?faces-redirect=true&amp;includeViewParams=true";
}
```

page2.xhtml

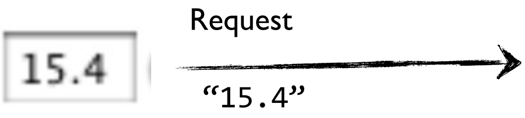


```
<f:metadata>
    <f:viewParam name="name" value="#{hello.name}"/>
</f:metadata>
<body>
    #{hello.name}
```



Conversion

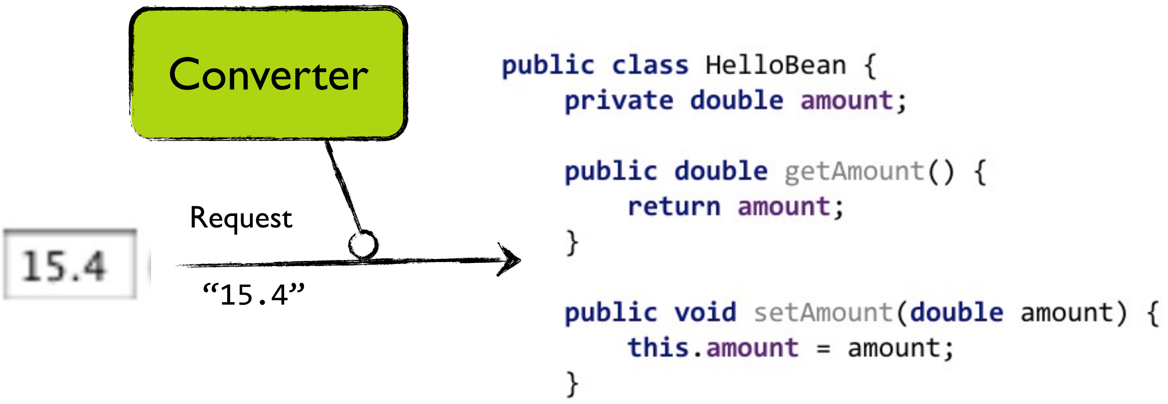
- ☐ Everything in a request is a String
- ☐ Convert request params to specific type
- ☐ Create String from specific type for rendering



```
public class HelloBean {
    private double amount;

    public double getAmount() {
        return amount;
    }

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

Conversion

- Everything in a request is a String
- Convert request params to specific type
- Create String from specific type for rendering



```
public class HelloBean {
    private double amount;

    public double getAmount() {
        return amount;
    }

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

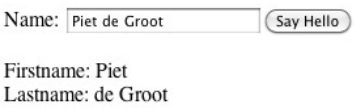
Default converters

- No explicit converter configuration needed for standard types

javax.faces.convert.BigDecimalConverter
javax.faces.convert.BigIntegerConverter
javax.faces.convert.BooleanConverter
javax.faces.convert.ByteConverter
javax.faces.convert.CharacterConverter
javax.faces.convert.DateTimeConverter
javax.faces.convert.DoubleConverter
javax.faces.convert.FloatConverter

Custom Converters

- ❑ Use custom types on managed bean



Name: Piet de Groot Say Hello

Firstname: Piet
Lastname: de Groot

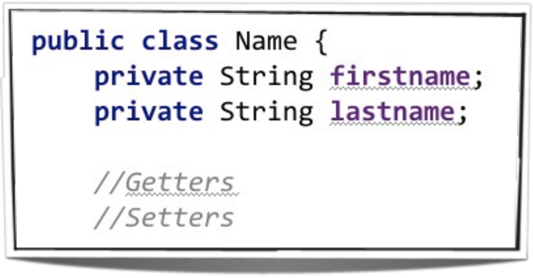
Request

“Piet de Groot”

```
public class HelloBean {
    private Name name;

    public Name getName() {
        return name;
    }

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

Custom Converters (2)

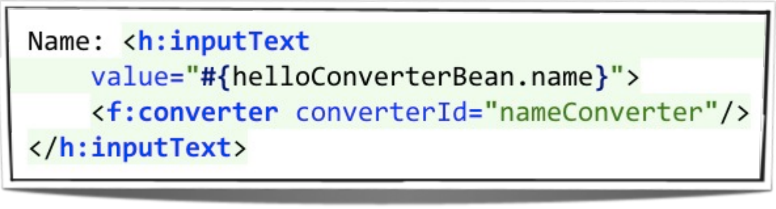
The complex type “Name”



```
public class Name {
    private String firstname;
    private String lastname;

    //Getters
    //Setters
```

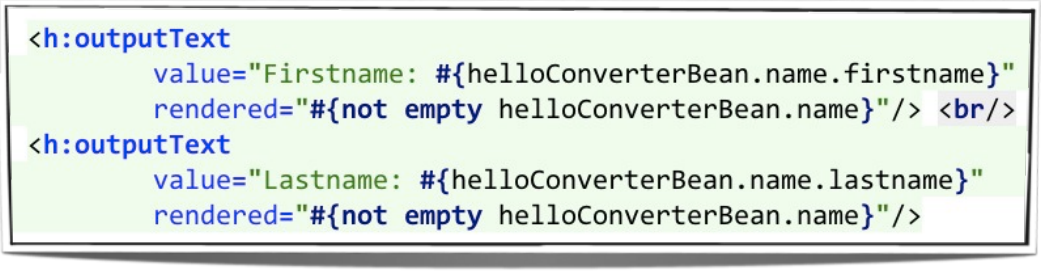
Custom Converters (3)

Input



```
Name: <h:inputText
    value="#{helloConverterBean.name}">
    <f:converter converterId="nameConverter"/>
</h:inputText>
```

Output

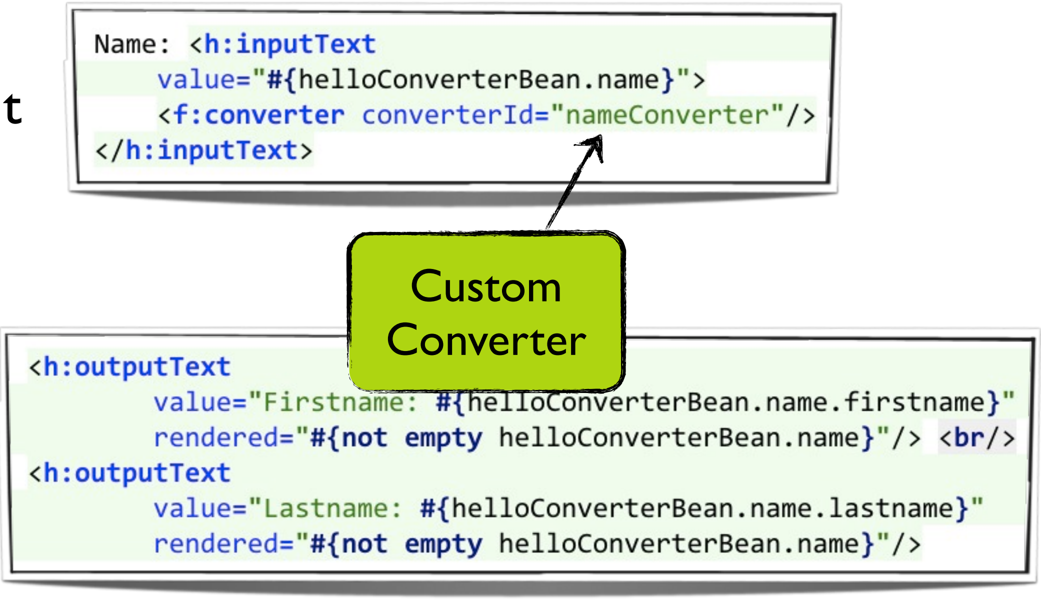


```
<h:outputText
        value="Firstname: #{helloConverterBean.name.firstname}"
        rendered="#{not empty helloConverterBean.name}"/> <br/>
<h:outputText
        value="Lastname: #{helloConverterBean.name.lastname}"
        rendered="#{not empty helloConverterBean.name}"/>
```

Custom Converters (3)

Input

Output



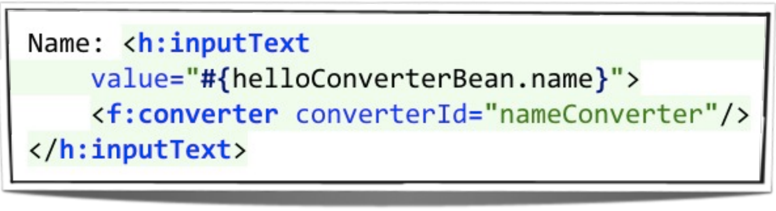
```
Name: <h:inputText
    value="#{helloConverterBean.name}">
    <f:converter converterId="nameConverter"/>
</h:inputText>
```

Custom Converter

```
<h:outputText
        value="Firstname: #{helloConverterBean.name.firstname}"
        rendered="#{not empty helloConverterBean.name}"/> <br/>
<h:outputText
        value="Lastname: #{helloConverterBean.name.lastname}"
        rendered="#{not empty helloConverterBean.name}"/>
```

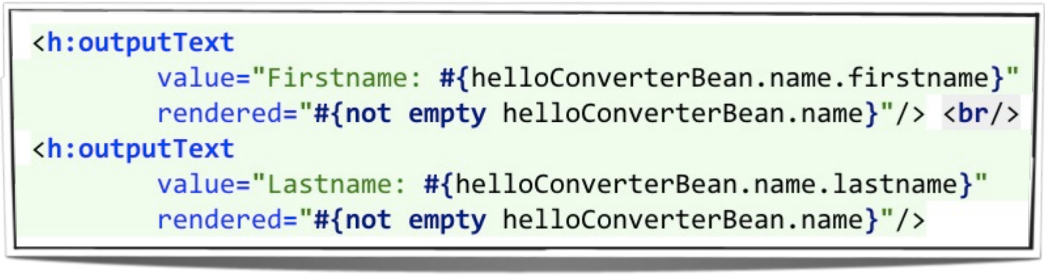
Custom Converters (3)

Input



```
Name: <h:inputText
    value="#{helloConverterBean.name}">
    <f:converter converterId="nameConverter"/>
</h:inputText>
```

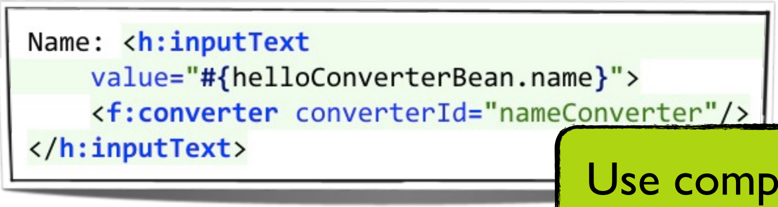
Output



```
<h:outputText
        value="Firstname: #{helloConverterBean.name.firstname}"
        rendered="#{not empty helloConverterBean.name}"/> <br/>
<h:outputText
        value="Lastname: #{helloConverterBean.name.lastname}"
        rendered="#{not empty helloConverterBean.name}"/>
```

Custom Converters (3)

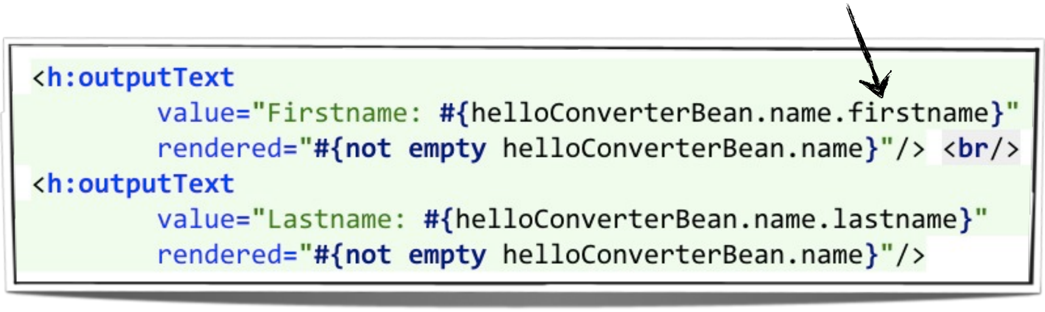
Input



```
Name: <h:inputText
    value="#{helloConverterBean.name}">
    <f:converter converterId="nameConverter"/>
</h:inputText>
```

Use complex type

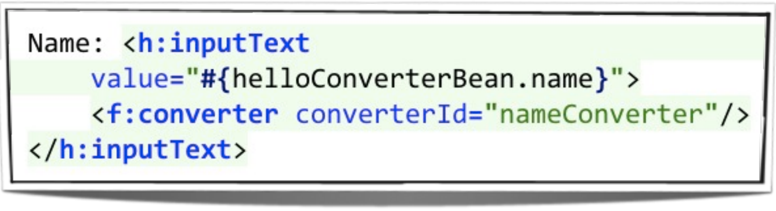
Output



```
<h:outputText
        value="Firstname: #{helloConverterBean.name.firstname}"
        rendered="#{not empty helloConverterBean.name}"/> <br/>
<h:outputText
        value="Lastname: #{helloConverterBean.name.lastname}"
        rendered="#{not empty helloConverterBean.name}"/>
```

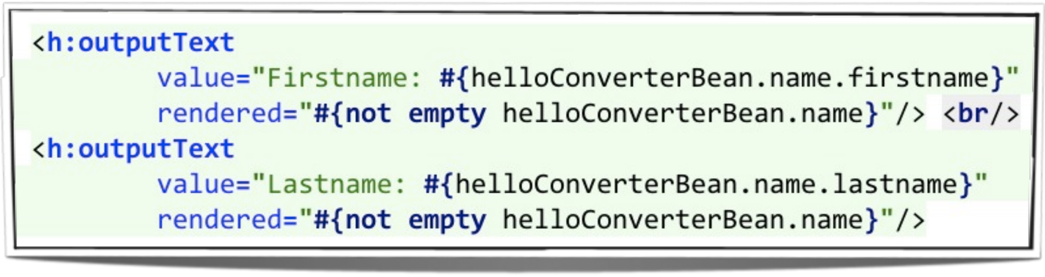
Custom Converters (3)

Input



```
Name: <h:inputText
        value="#{helloConverterBean.name}">
        <f:converter converterId="nameConverter"/>
</h:inputText>
```

Output

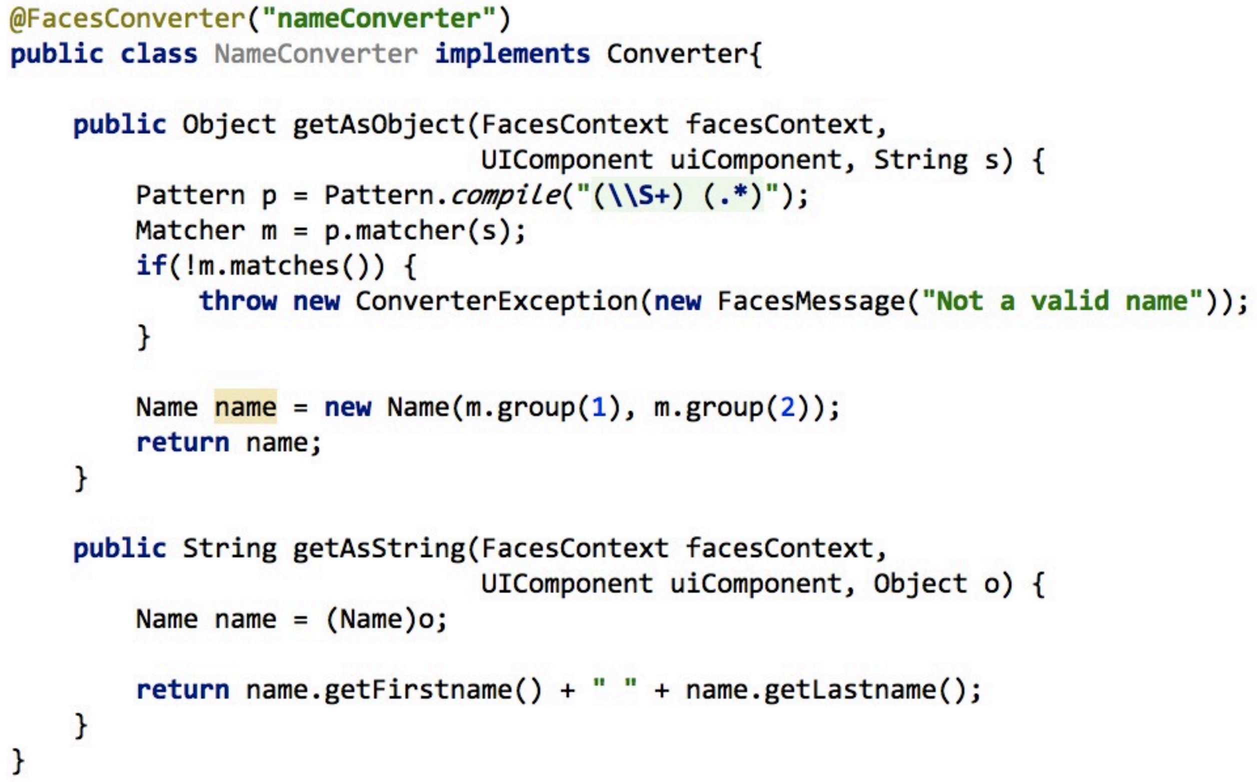


```
<h:outputText
        value="Firstname: #{helloConverterBean.name.firstname}"
        rendered="#{not empty helloConverterBean.name}"/> <br/>
<h:outputText
        value="Lastname: #{helloConverterBean.name.lastname}"
        rendered="#{not empty helloConverterBean.name}"/>
```

Writing a converter

- ❑ Implement the javax.faces.Converter interface
  - ○ getAsObject (from String to object)
  - ○ getAsString (from object to String)
- ❑ Annotate @FacesConverter
- ❑ Throw ConverterException if conversion fails

NameConverter

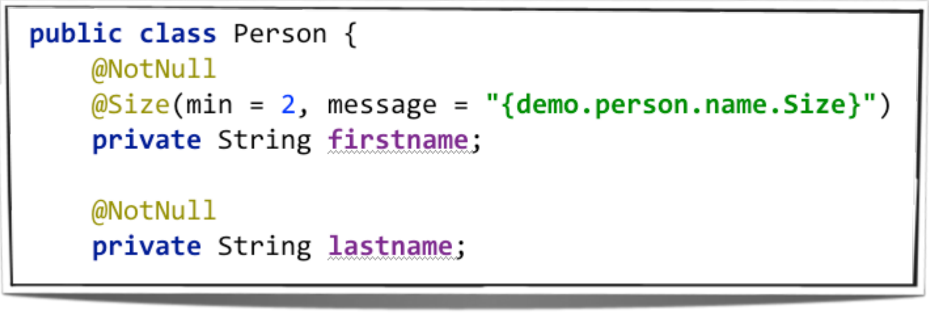


```
@FacesConverter("nameConverter")
public class NameConverter implements Converter{

    public Object getAsObject(FacesContext facesContext,
                              UIComponent uiComponent, String s) {
        Pattern p = Pattern.compile("(\\S+) (.*)");
        Matcher m = p.matcher(s);
        if(!m.matches()) {
            throw new ConverterException(new FacesMessage("Not a valid name"));
        }

        Name name = new Name(m.group(1), m.group(2));
        return name;
    }

    public String getAsString(FacesContext facesContext,
                              UIComponent uiComponent, Object o) {
        Name name = (Name)o;

        return name.getFirstname() + " " + name.getLastname();
    }
}
```

Validation

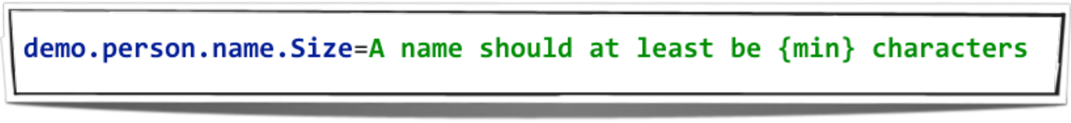
- Validate if an input value is acceptable
  - e.g. minimum of 3 characters
  - between 18 and 65
- Validation happens after conversion
- JSF integrates with Bean Validation

Specifying constraints



```
public class Person {
    @NotNull
    @Size(min = 2, message = "{demo.person.name.Size}")
    private String firstname;

    @NotNull
    private String lastname;
```

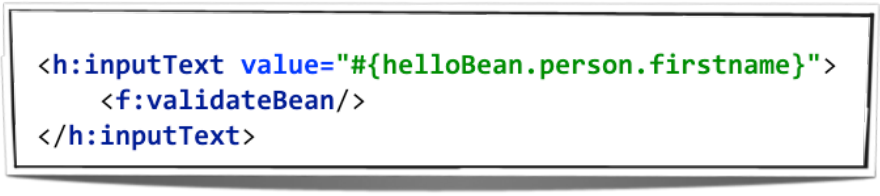
ValidationMessages.properties



```
demo.person.name.Size=A name should at least be {min} characters
```

Validating fields

- □ No explicit required=”true” required to validate empty fields
- □ Validation triggered automatically
- □ Only fields that are on the page are validated

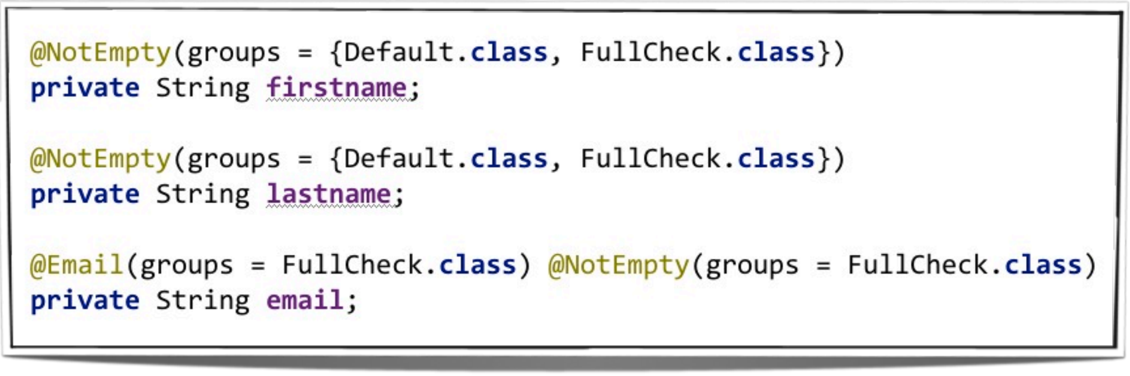


```
<h:inputText value="#{helloBean.person.firstname}">
    <f:validateBean/>
</h:inputText>
```

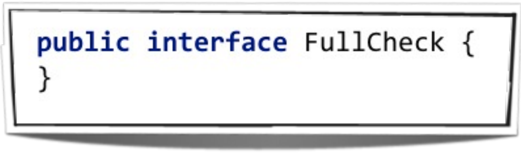
Validation Groups

- Sometimes only part of the constraints should be checked
  - e.g. in a wizard
- Can be achieved using Validation Groups
- Not often necessary in JSF
  - only fields on a page are validated

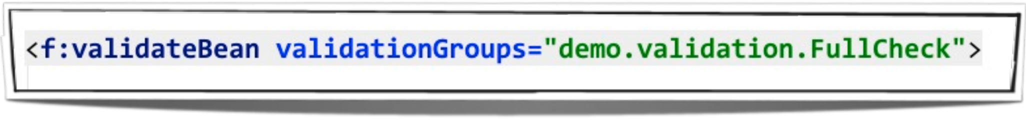
Validation Groups



```java
@NotEmpty(groups = {Default.class, FullCheck.class})
private String firstname;

@NotEmpty(groups = {Default.class, FullCheck.class})
private String lastname;

@Email(groups = FullCheck.class) @NotEmpty(groups = FullCheck.class)
private String email;
```

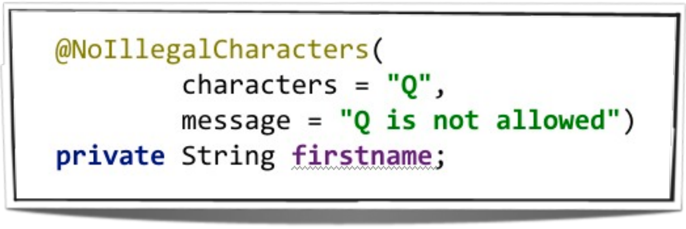


```java
public interface FullCheck {
}
```

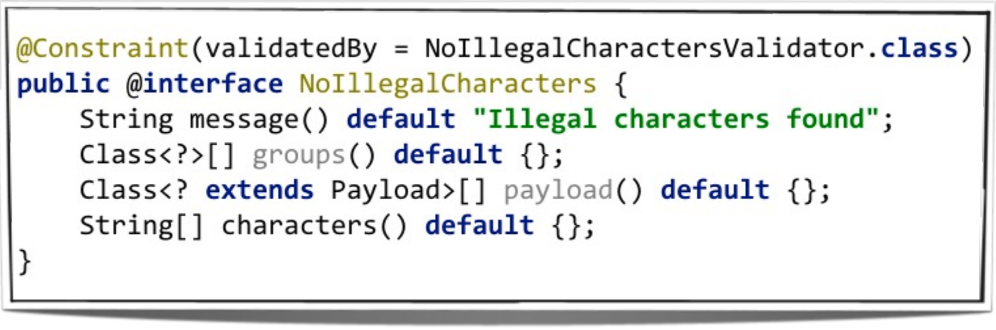


```xml
<f:validateBean validationGroups="demo.validation.FullCheck">
```

Custom validators

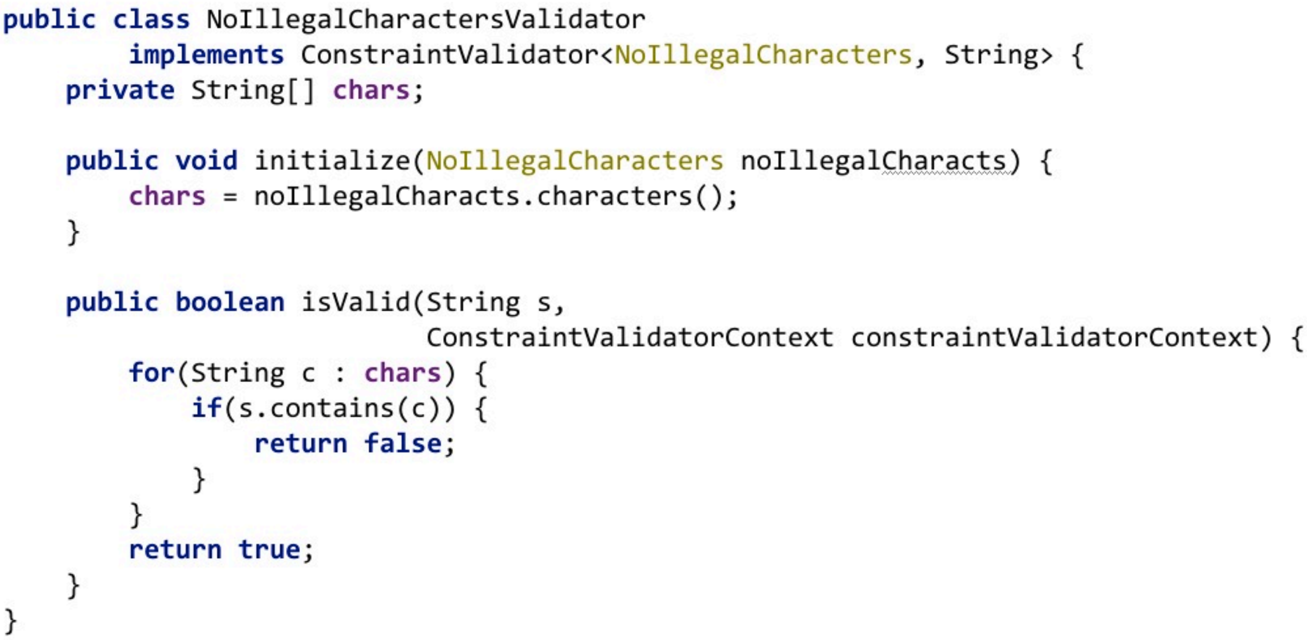


```
@NoIllegalCharacters(
        characters = "Q",
        message = "Q is not allowed")
private String firstname;
```



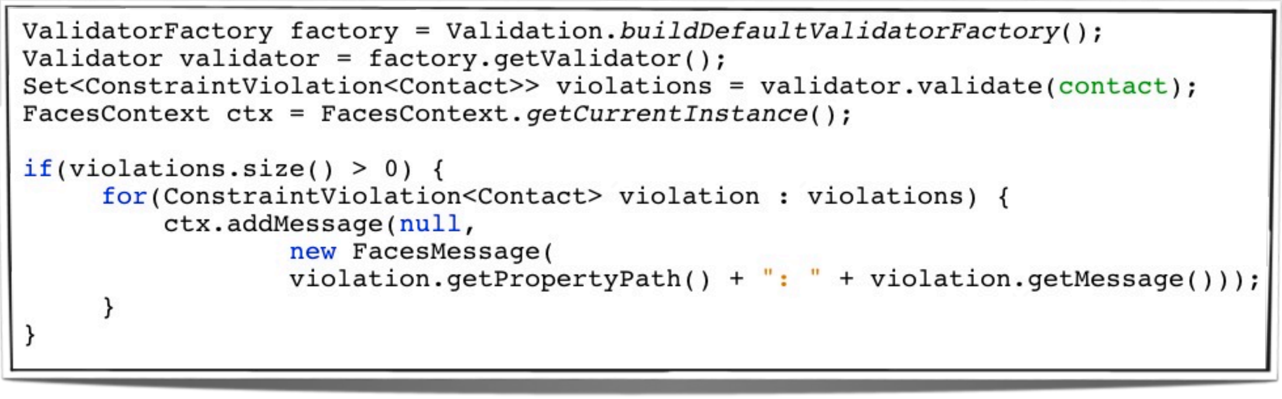
```
@Constraint(validatedBy = NoIllegalCharactersValidator.class)
public @interface NoIllegalCharacters {
    String message() default "Illegal characters found";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
    String[] characters() default {};
}
```

Validator implementation



```
public class NoIllegalCharactersValidator
        implements ConstraintValidator<NoIllegalCharacters, String> {
    private String[] chars;

    public void initialize(NoIllegalCharacters noIllegalCharacts) {
        chars = noIllegalCharacts.characters();
    }

    public boolean isValid(String s,
                           ConstraintValidatorContext constraintValidatorContext) {
        for(String c : chars) {
            if(s.contains(c)) {
                return false;
            }
        }
        return true;
    }
}
```

Programmatic validation



```
ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
Validator validator = factory.getValidator();
Set<ConstraintViolation<Contact>> violations = validator.validate(contact);
FacesContext ctx = FacesContext.getCurrentInstance();

if(violations.size() > 0) {
     for(ConstraintViolation<Contact> violation : violations) {
         ctx.addMessage(null,
                 new FacesMessage(
                 violation.getPropertyPath() + ": " + violation.getMessage()));
     }
}
```

Multi-field validation

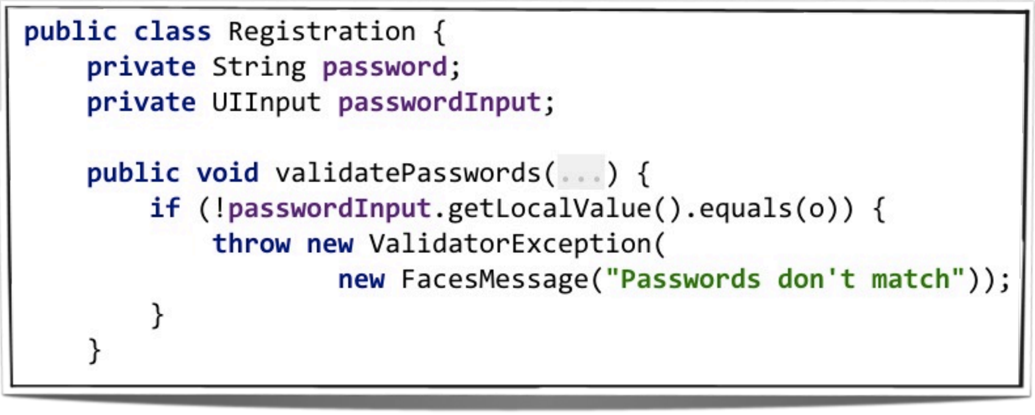
- JSF validation is always about a single component
- Sometimes you need validation over multiple fields
  - e.g. repeat password

Multi-field validation (2)

- Multiple solutions
  - Use local values in validator mechanism (values are not set)
  - validate in action method (values are set) <u>not recommended</u>

Using a validator method

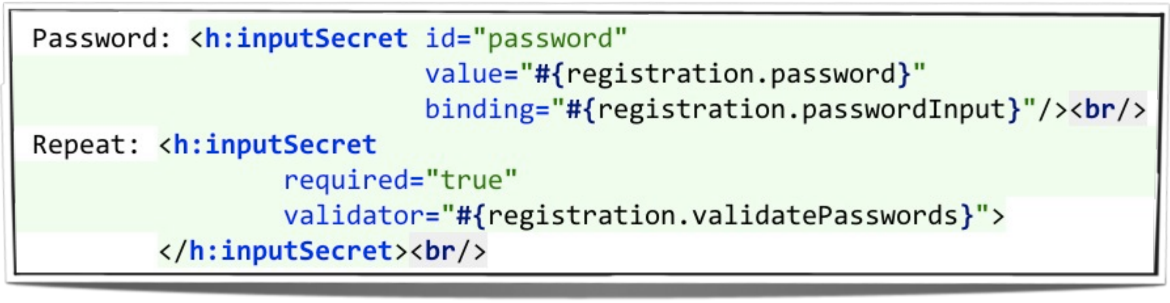
- No lookup required



```
public class Registration {
    private String password;
    private UIInput passwordInput;

    public void validatePasswords(...) {
        if (!passwordInput.getLocalValue().equals(o)) {
            throw new ValidatorException(
                    new FacesMessage("Passwords don't match"));
        }
    }
```

Using a validator method (2)

- Need to set the binding property

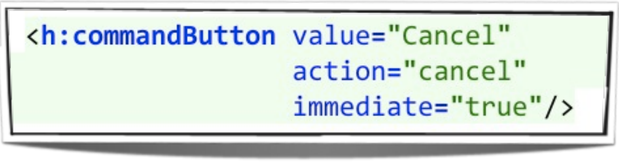


```
Password: <h:inputSecret id="password"
                         value="#{registration.password}"
                         binding="#{registration.passwordInput}"/><br/>
Repeat: <h:inputSecret
                required="true"
                validator="#{registration.validatePasswords}">
        </h:inputSecret><br/>
```

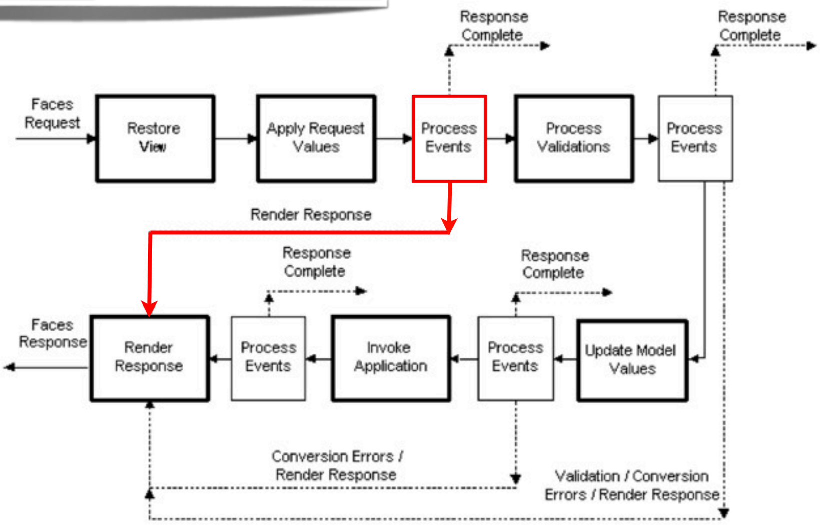
Immediate

- Sometimes conversion and validation should be skipped to execute an action
  - e.g. a cancel button
- Use the immediate property
  - executes action after Apply Request Values phase
  - calls Render Response phase

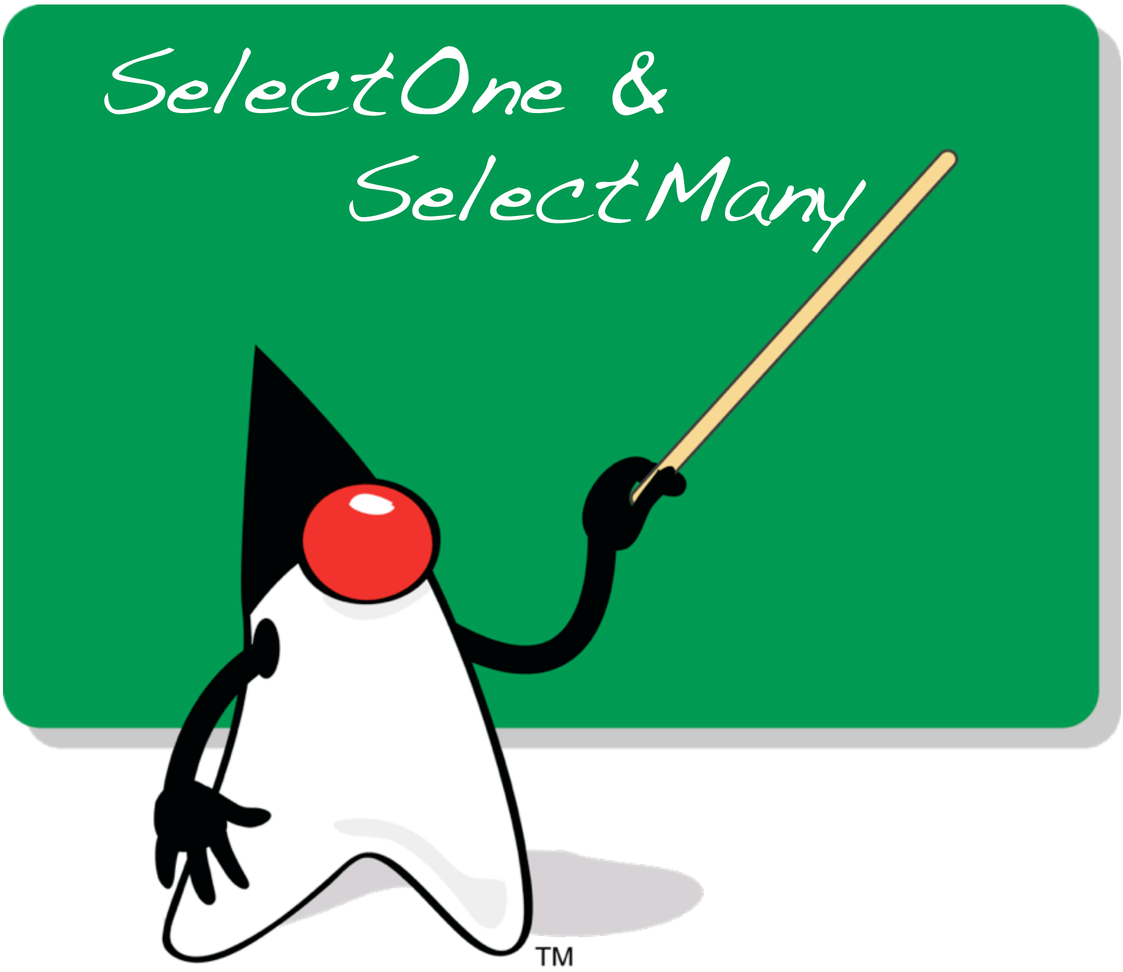
Immediate example



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



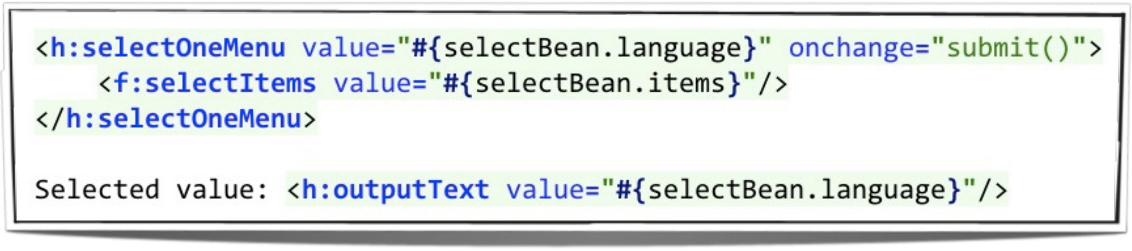
SelectOne & SelectMany



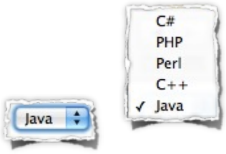
Select, radio and check boxes

- Different ways to render choices
  - either single or multiple selection possible
- Items rendered from SelectItem
  - can be created from a Map
- Value is the selected item(s)

h:selectOneListbox



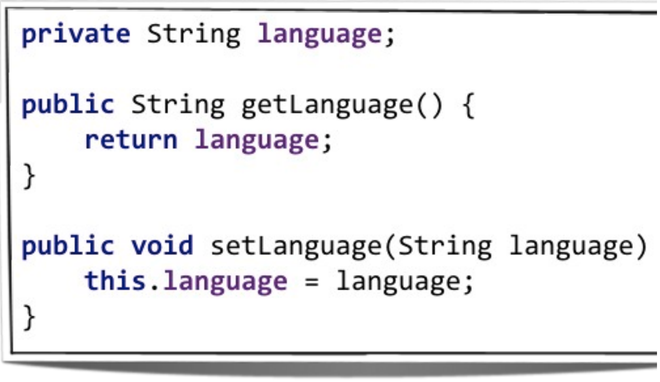
```
<h:selectOneMenu value="#{selectBean.language}" onchange="submit()">
    <f:selectItems value="#{selectBean.items}"/>
</h:selectOneMenu>

Selected value: <h:outputText value="#{selectBean.language}"/>
```



h:selectOneListbox

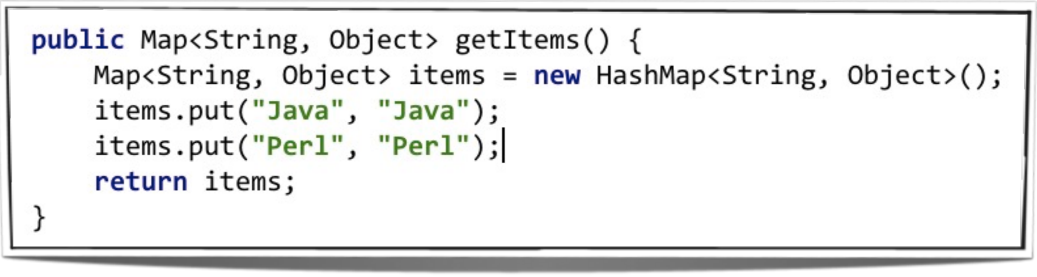
Selected item

Type must be the Map's value type



```
private String language;

public String getLanguage() {
    return language;
}

public void setLanguage(String language)
    this.language = language;
}
```

Items from map



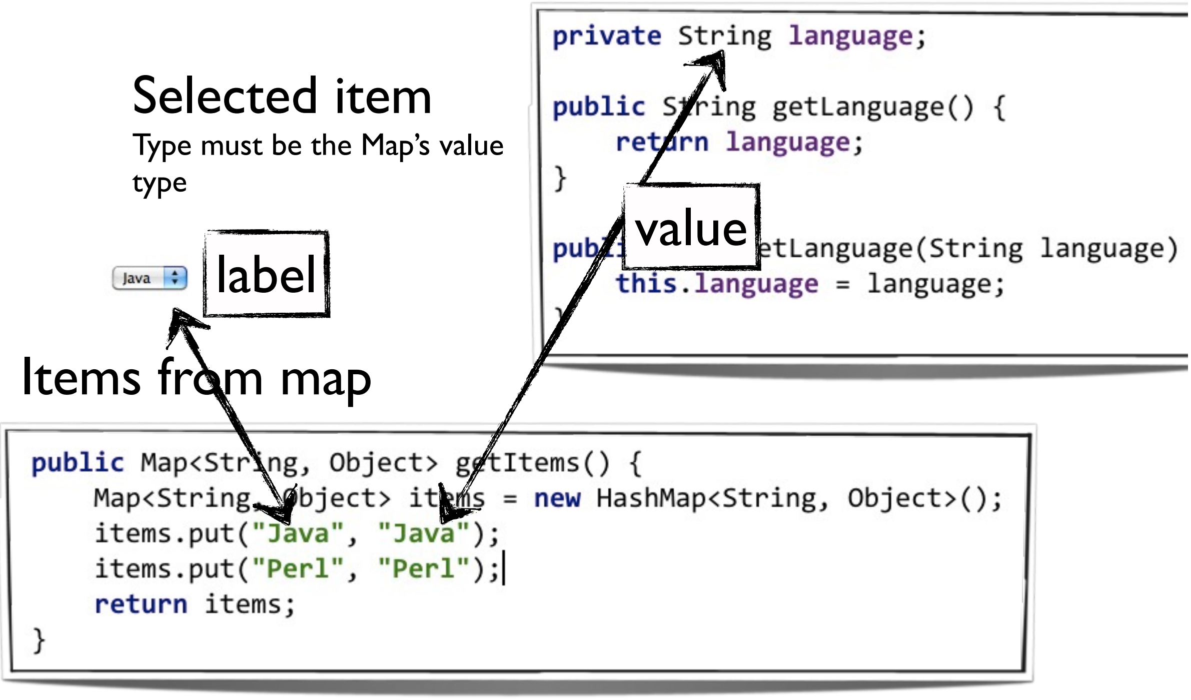
```
public Map<String, Object> getItems() {
    Map<String, Object> items = new HashMap<String, Object>();
    items.put("Java", "Java");
    items.put("Perl", "Perl");
    return items;
}
```

h:selectOneListbox

Selected item

Type must be the Map's value type

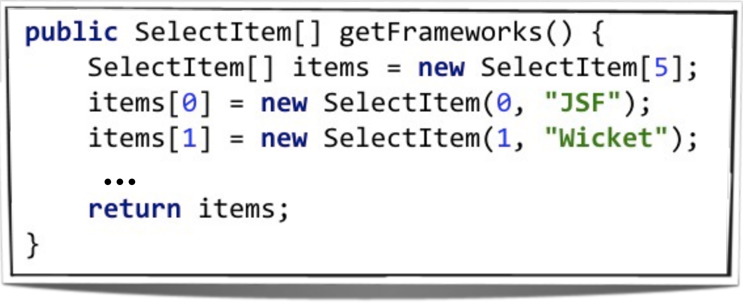
Items from map



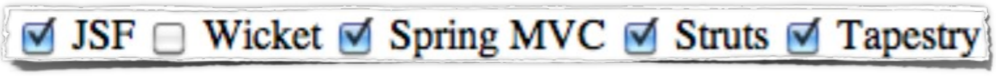
```
private String language;

public String getLanguage() {
    return language;
}

public void setLanguage(String language)
    this.language = language;
}
```

```
public Map<String, Object> getItems() {
    Map<String, Object> items = new HashMap<String, Object>();
    items.put("Java", "Java");
    items.put("Perl", "Perl");
    return items;
}
```

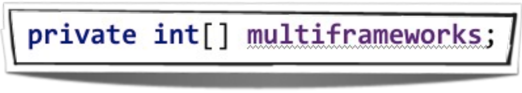
h:selectManyCheckbox



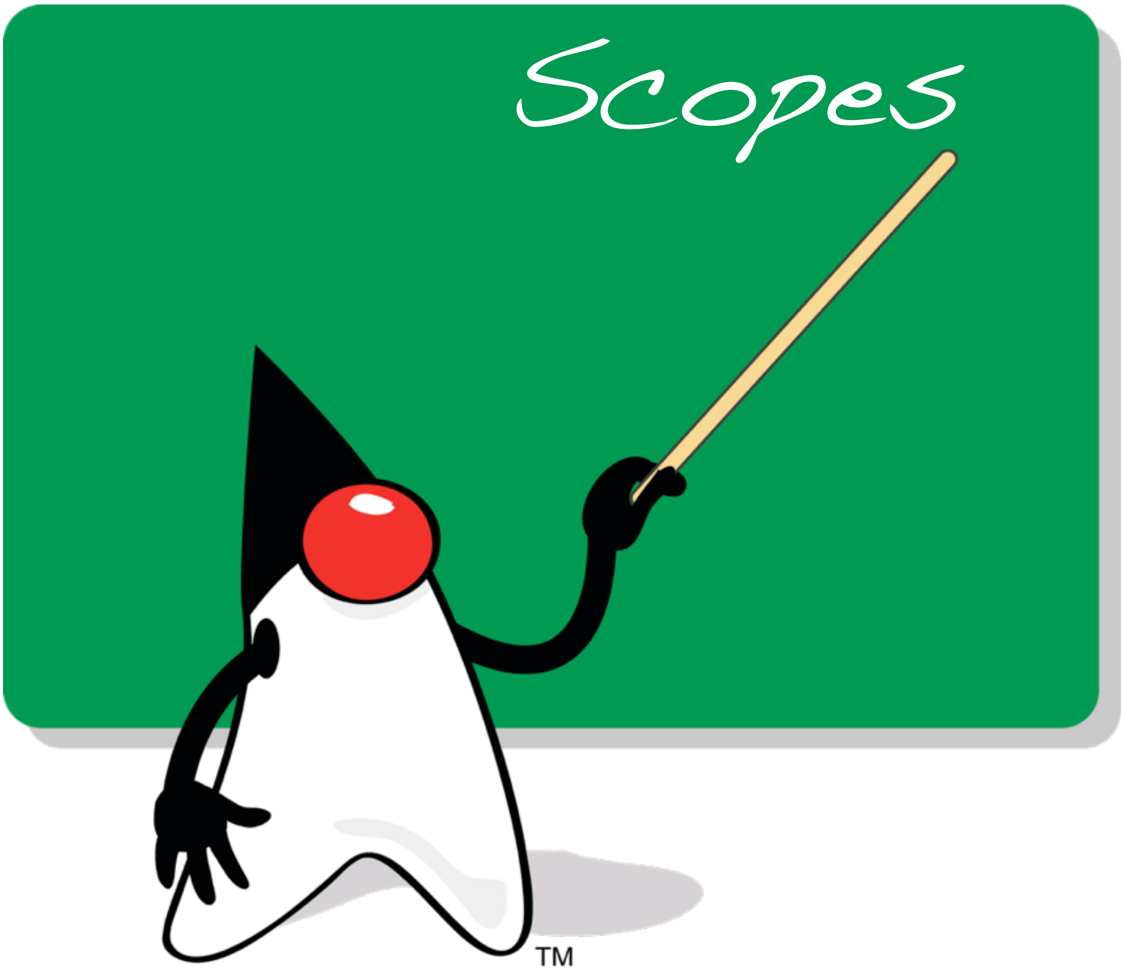
```
public SelectItem[] getFrameworks() {
    SelectItem[] items = new SelectItem[5];
    items[0] = new SelectItem(0, "JSF");
    items[1] = new SelectItem(1, "Wicket");
     ...
    return items;
}
```



☑ JSF □ Wicket ☑ Spring MVC ☑ Struts ☑ Tapestry



```
private int[] multiframeworks;
```



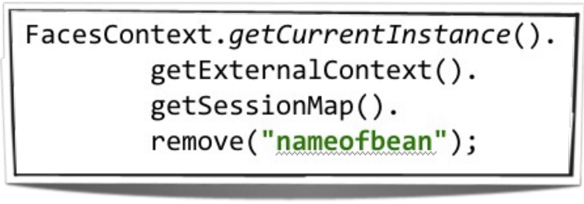
Scopes

State

- A managed bean has a scope
  - request, session or application
- Prefer request scope
  - Not possible for multi request/page scenarios
  - Not possible after redirect
- Don’t forget to clean-up session scope

Remove bean from session

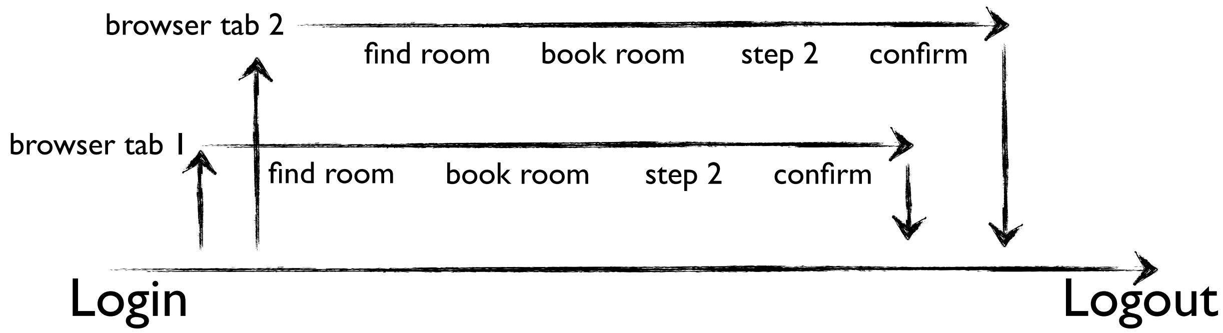
- Session scoped beans stay in the session until the session is destroyed
  - Potential high memory footprint
- Remove beans from sessions when possible



```
FacesContext.getCurrentInstance().
        getExternalContext().
        getSessionMap().
        remove("nameofbean");
```

Conversation scope

- Multi-request scope *within* a session
- Isolated from other conversations
- Defined begin and end point



browser tab 2

find room book room step 2 confirm

browser tab 1

find room book room step 2 confirm

Login

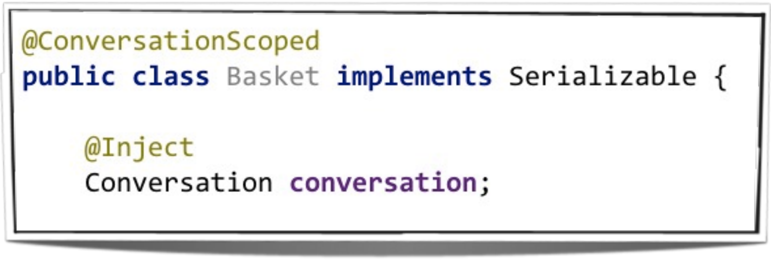
Logout

Conversations

- Each JSF request has an active conversation
  - transient by default - destroyed at the end of the request
- Can be upgraded to long-running

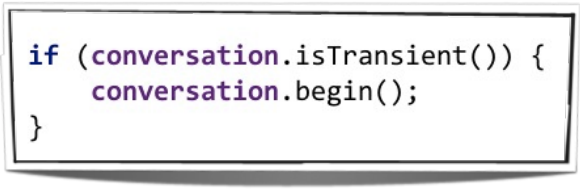
Long running conversation

Conversational bean



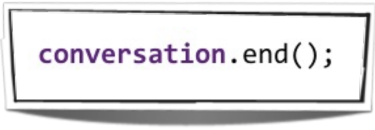
```
@ConversationScoped
public class Basket implements Serializable {

    @Inject
    Conversation conversation;
```

Upgrade to long-running

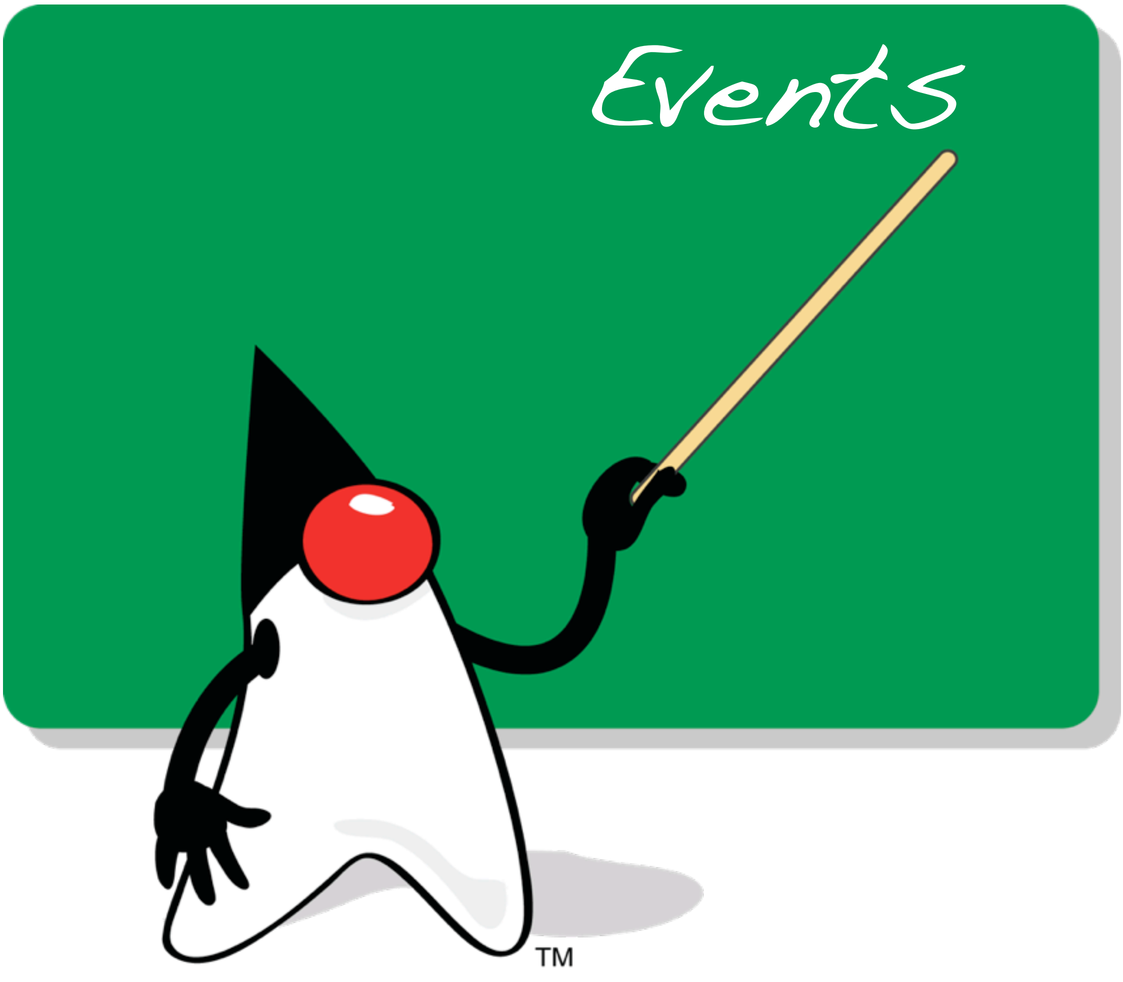


```
if (conversation.isTransient()) {
    conversation.begin();
}
```

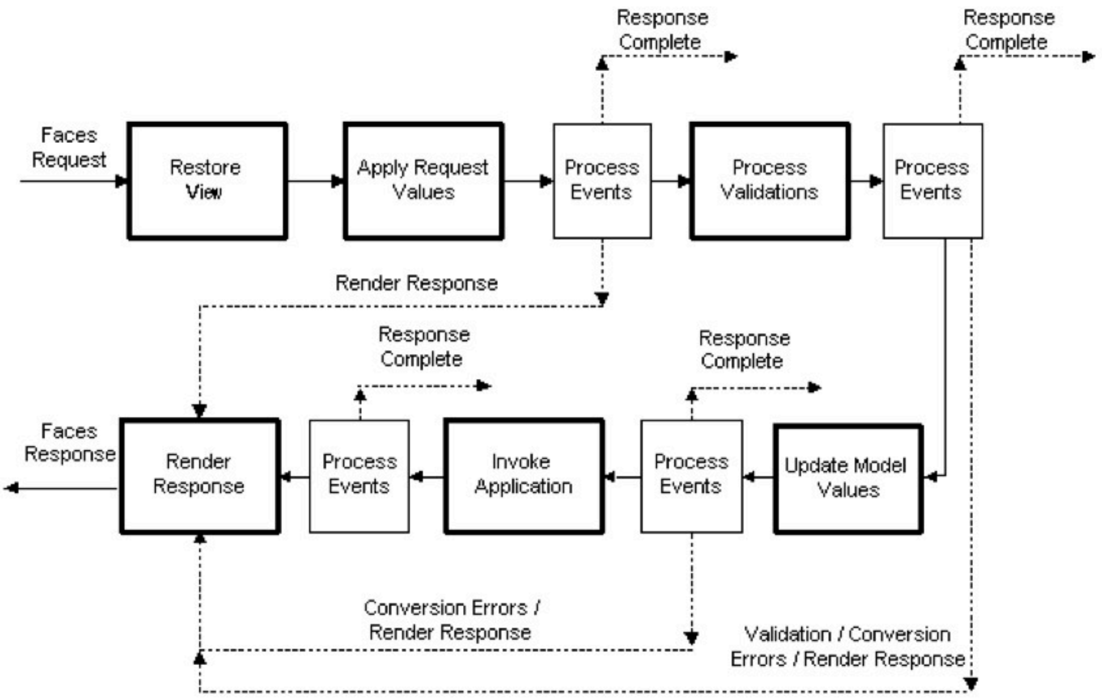
Schedule end of conversation at end of request



```
conversation.end();
```



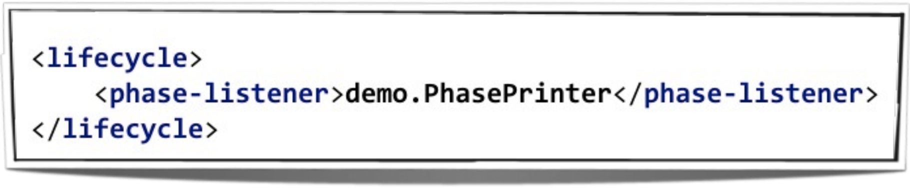
The life cycle



PhaseListener

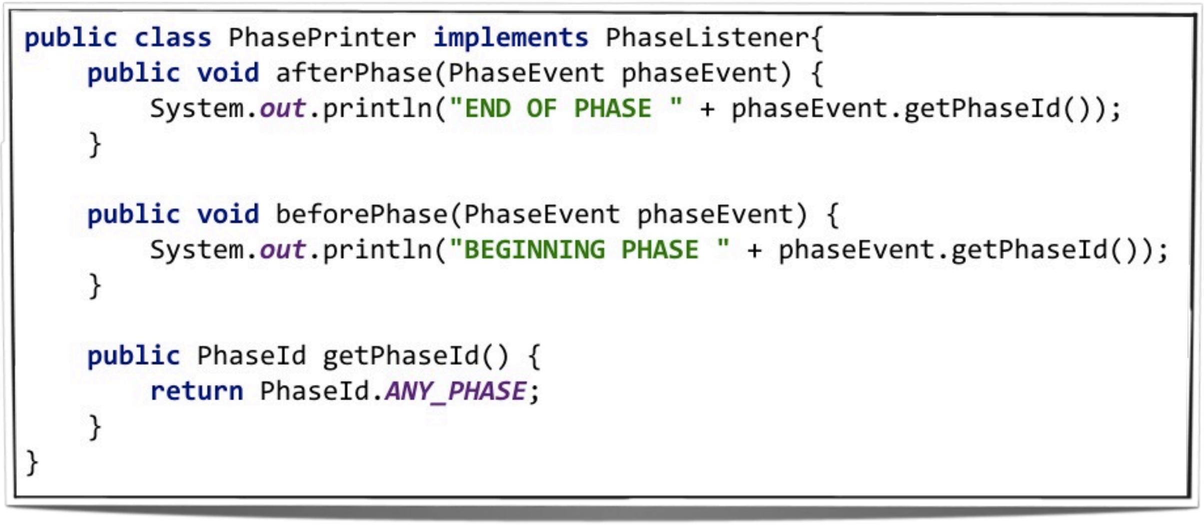
- Execute code before/after a certain phase
- Very useful for debugging and understanding the life-cycle

Register in faces-config.xml



```
<lifecycle>
    <phase-listener>demo.PhasePrinter</phase-listener>
</lifecycle>
```

PhaseListener example

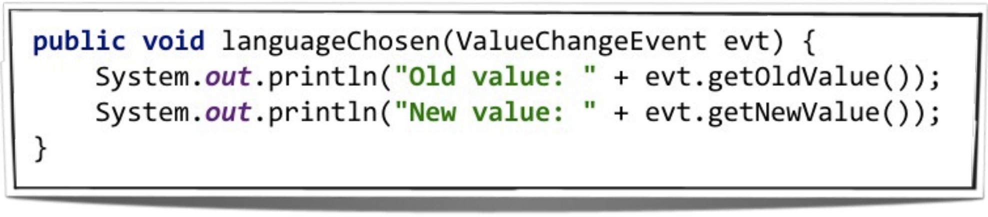


```java
public class PhasePrinter implements PhaseListener{
    public void afterPhase(PhaseEvent phaseEvent) {
        System.out.println("END OF PHASE " + phaseEvent.getPhaseId());
    }

    public void beforePhase(PhaseEvent phaseEvent) {
        System.out.println("BEGINNING PHASE " + phaseEvent.getPhaseId());
    }

    public PhaseId getPhaseId() {
        return PhaseId.ANY_PHASE;
    }
}
```

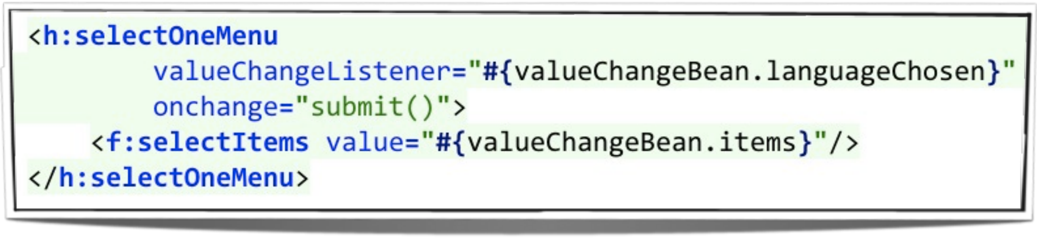
Value change events

- Possible on UIInput components
- Do not trigger a submit automatically
  - Use JavaScript onchange event
- Access to the new and old value
- Executed after *Process Validations* but before *Update Model*

Value change example

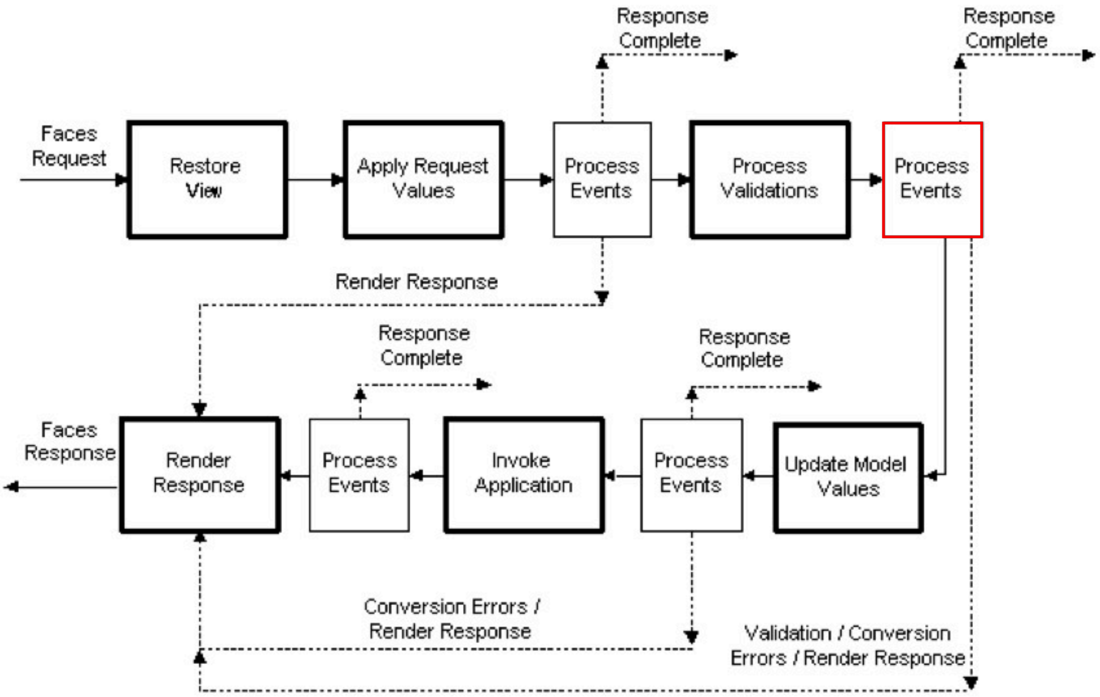


```java
public void languageChosen(ValueChangeEvent evt) {
    System.out.println("Old value: " + evt.getOldValue());
    System.out.println("New value: " + evt.getNewValue());
}
```



```xml
<h:selectOneMenu
        valueChangeListener="#{valueChangeBean.languageChosen}"
        onchange="submit()">
    <f:selectItems value="#{valueChangeBean.items}"/>
</h:selectOneMenu>
```

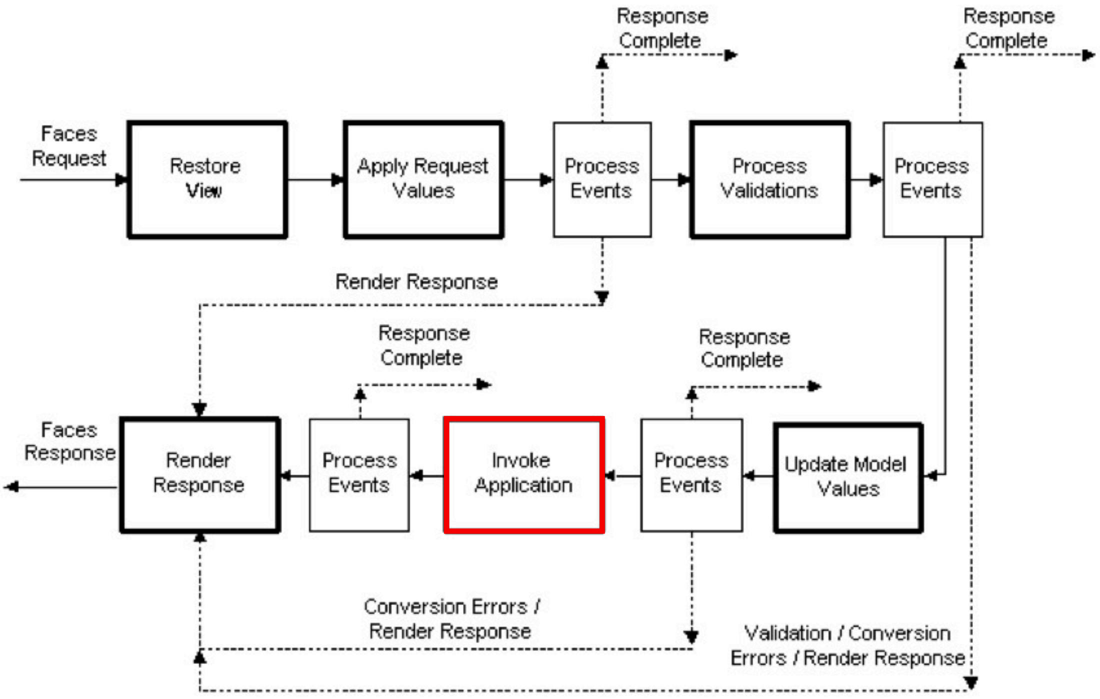
Value changed events



Action events

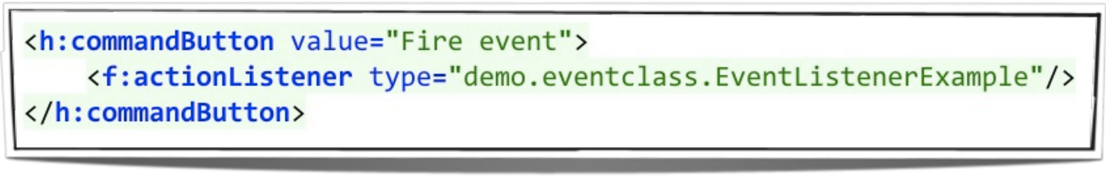
- Executed during *Invoke Application*
  - Before actions
- Distinguish actions and action events
  - actions -> business logic and navigation
  - action events -> UI related
  - action events know about the component that invoked the event

Action events



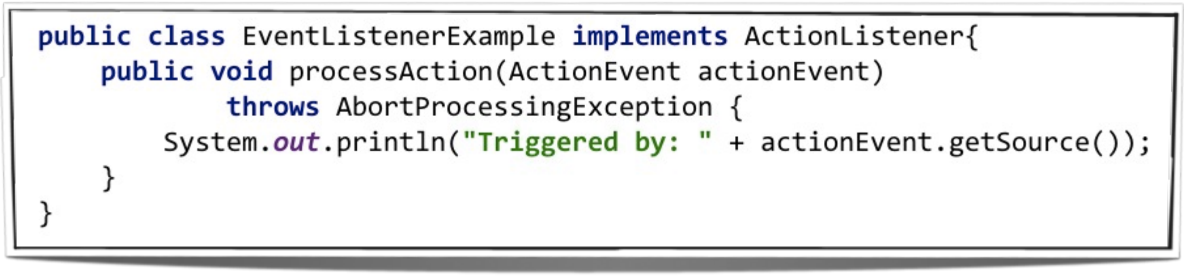
Event Listener Classes

- Class that implements ValueChangeListener or ActionEventListener
- Use f:valueChangeListener and f:actionListener tags
  - Allows multiple listeners

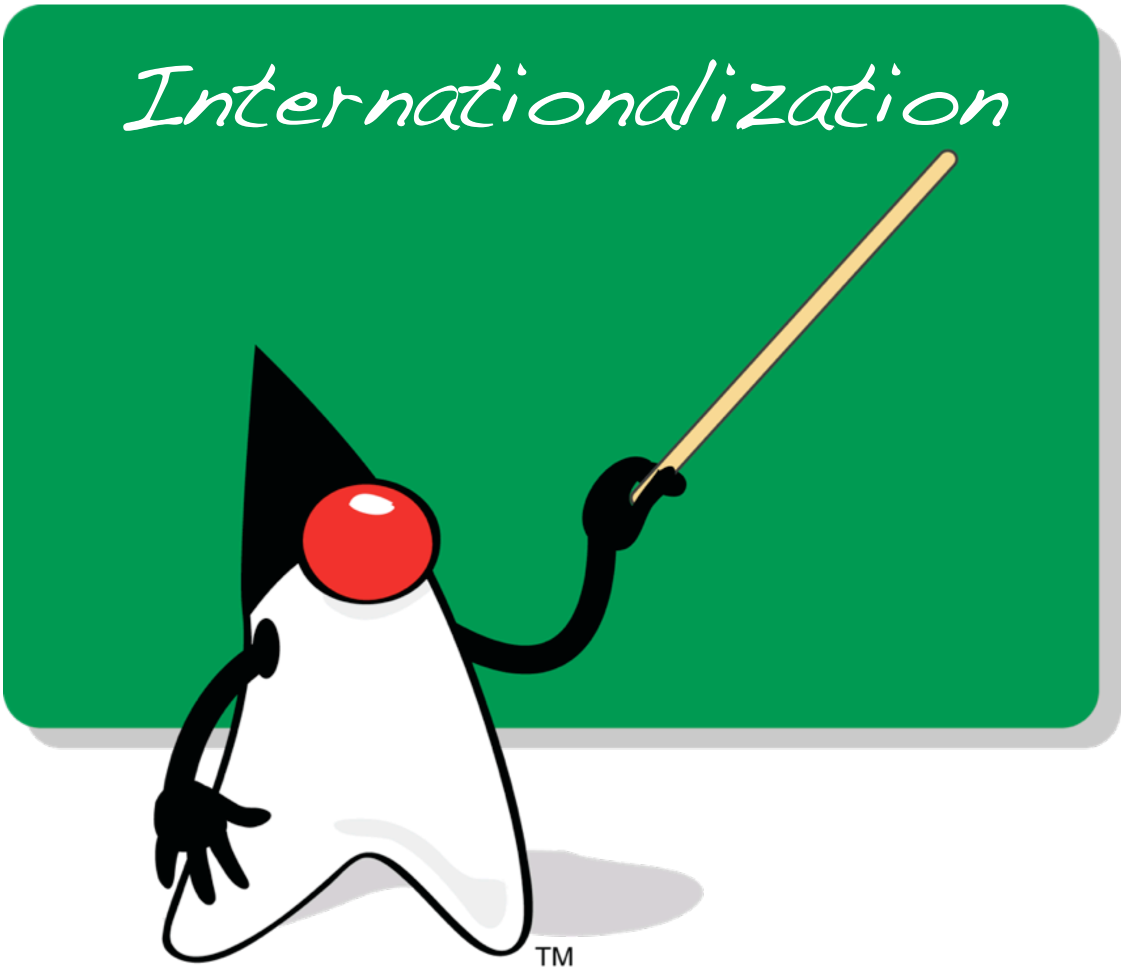


```
<h:commandButton value="Fire event">
    <f:actionListener type="demo.eventclass.EventListenerExample"/>
</h:commandButton>
```

Event Listener Classes



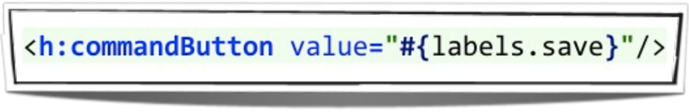
```
public class EventListenerExample implements ActionListener{
    public void processAction(ActionEvent actionEvent)
            throws AbortProcessingException {
        System.out.println("Triggered by: " + actionEvent.getSource());
    }
}
```



Internationalization

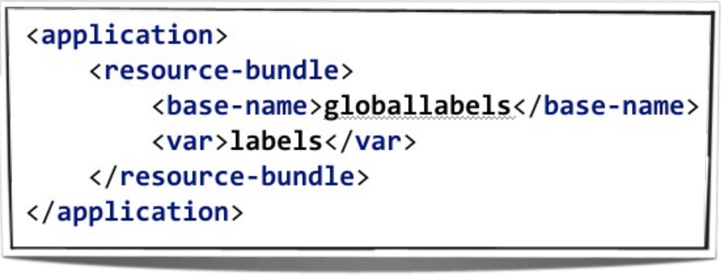
- Easy to load Strings from a message bundle
- A bundle is a properties file
- Support for localization
- Load bundle
  - in faces-config.xml
  - or import on page

Internationalization

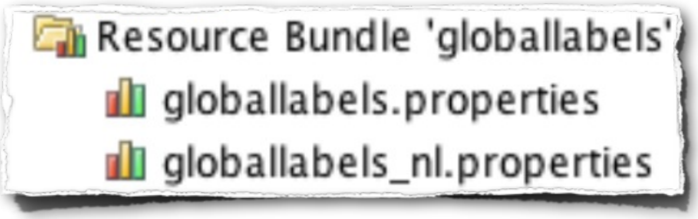


```
<h:commandButton value="#{labels.save}"/>
```

Available on every page



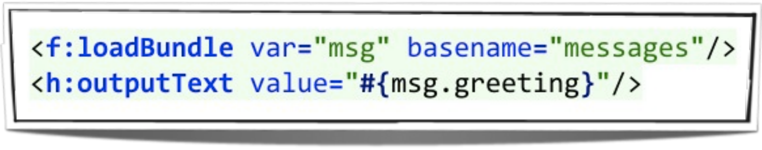
```
<application>
    <resource-bundle>
        <base-name>globallabels</base-name>
        <var>labels</var>
    </resource-bundle>
</application>
```



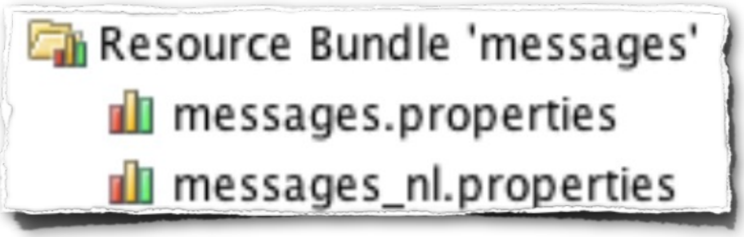
```
Resource Bundle 'globallabels'
    globallabels.properties
    globallabels_nl.properties
```

Internationalization

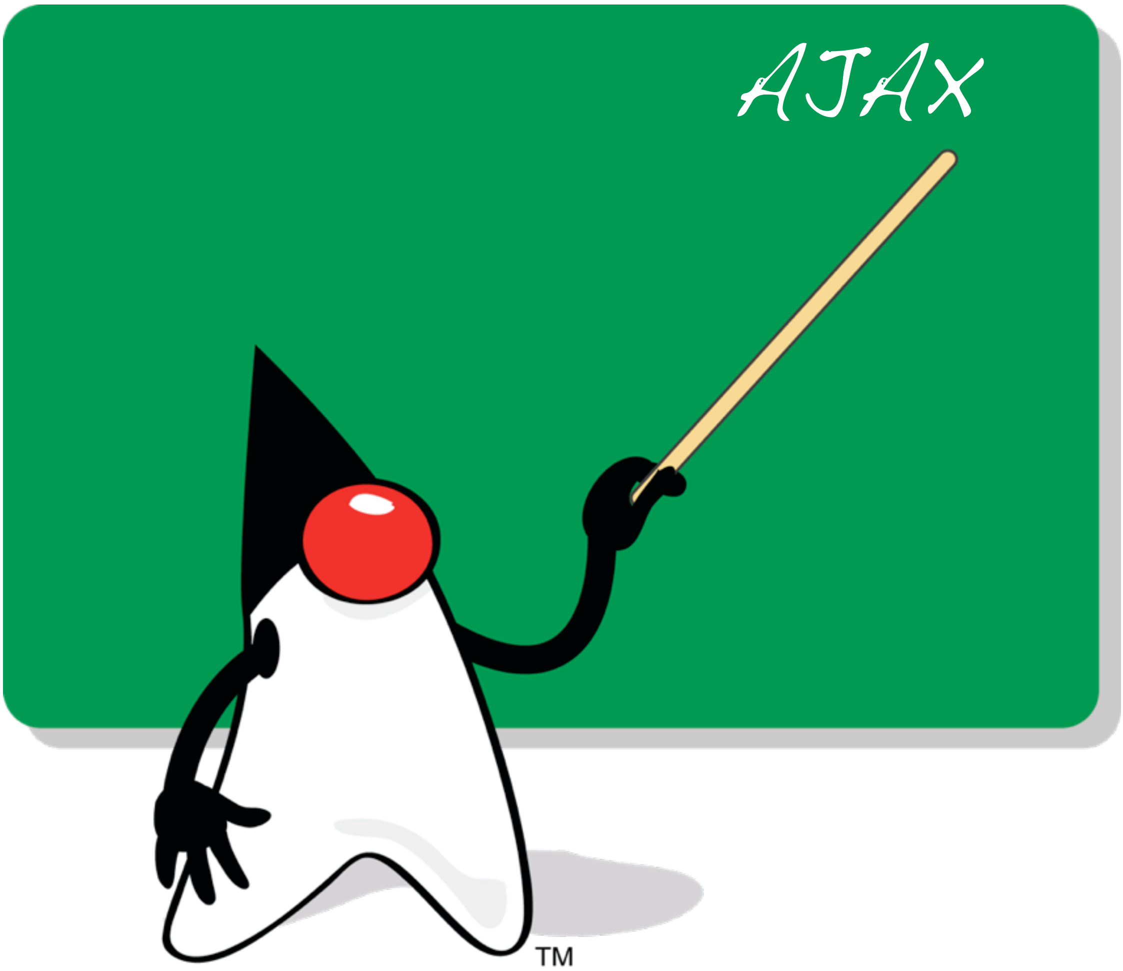
Load for a specific page



```
<f:loadBundle var="msg" basename="messages"/>
<h:outputText value="#{msg.greeting}"/>
```

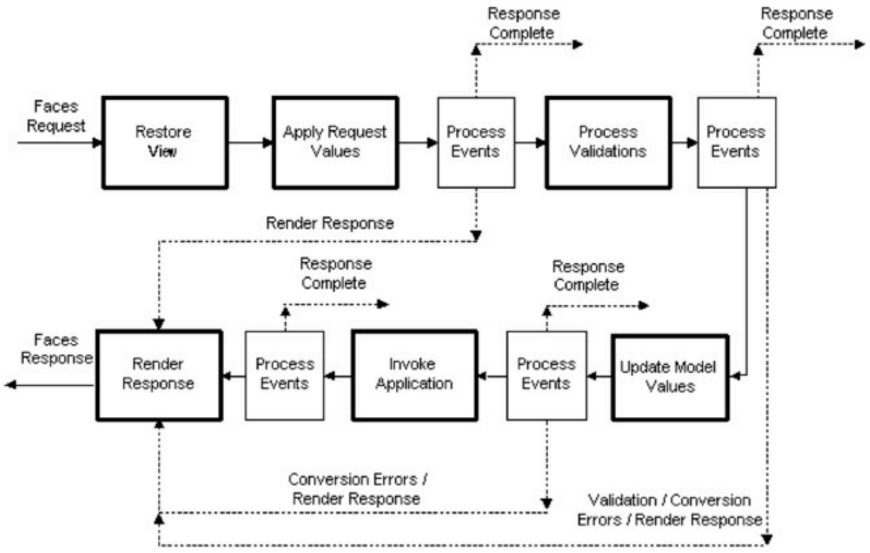


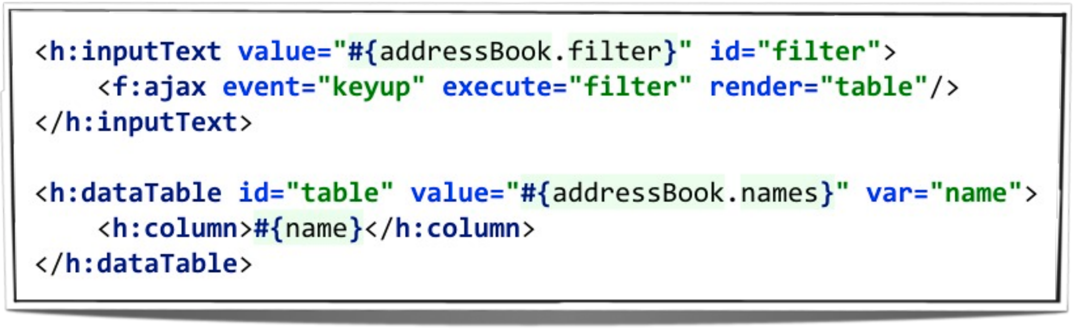
```
Resource Bundle 'messages'
    messages.properties
    messages_nl.properties
```



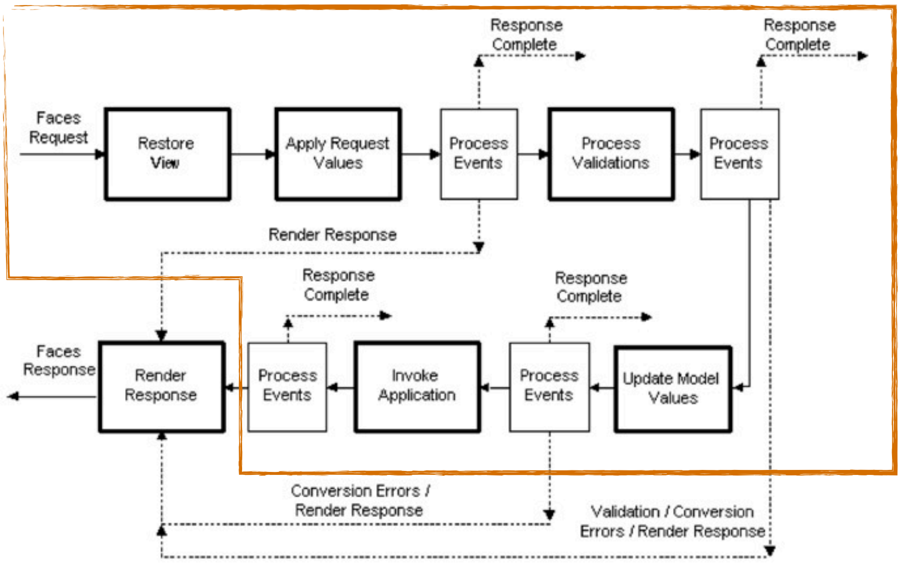
Ajax

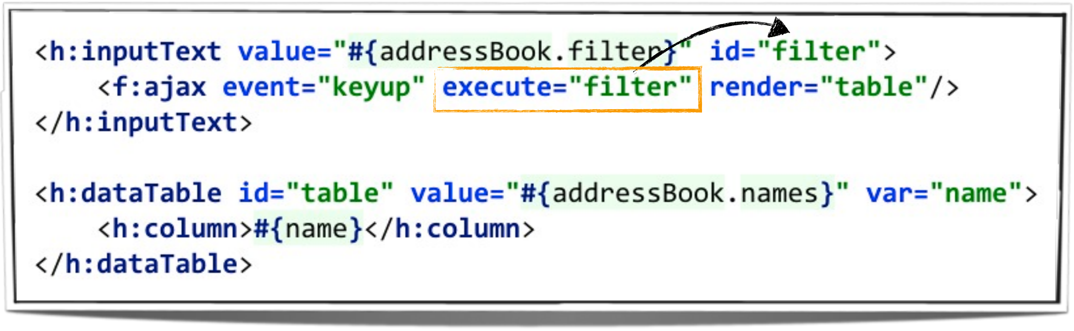
- ❑ Partial page loading made easy
  - ○ submit part of a page
  - ○ re-render part of a page
- ❑ Wrap f:ajax in an existing component
  - ○ specify event and render


Response Complete
Response Complete
Faces Request
Restore View
Apply Request Values
Process Events
Process Validations
Process Events
Render Response
Response Complete
Response Complete
Faces Response
Render Response
Process Events
Invoke Application
Process Events
Update Model Values
Conversion Errors / Render Response
Validation / Conversion Errors / Render Response

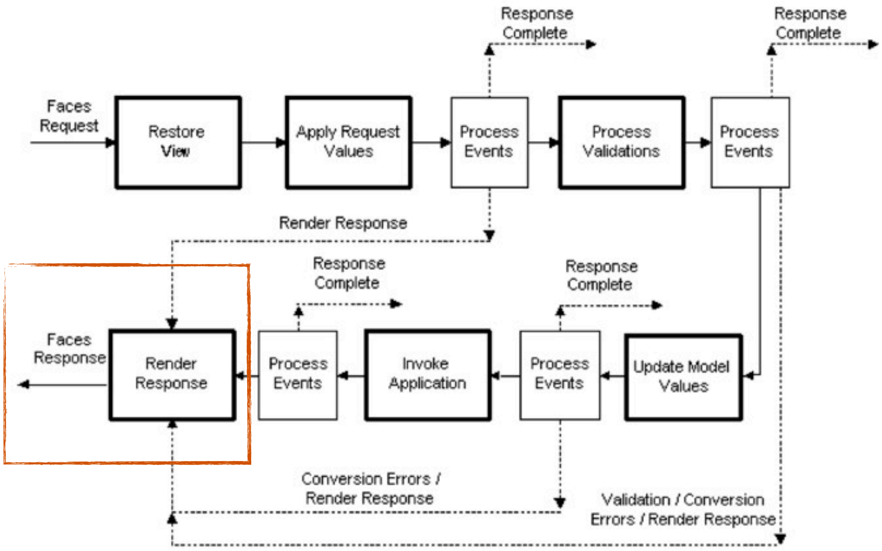


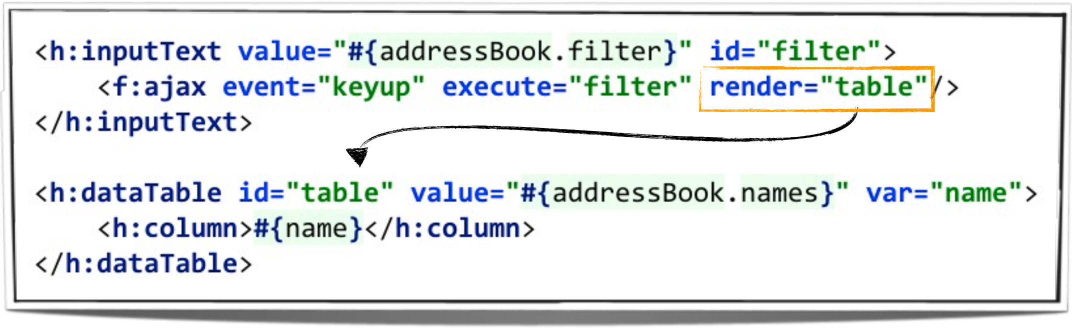
```
<h:inputText value="#{addressBook.filter}" id="filter">
    <f:ajax event="keyup" execute="filter" render="table"/>
</h:inputText>

<h:dataTable id="table" value="#{addressBook.names}" var="name">
    <h:column>#{name}</h:column>
</h:dataTable>
```


Response Complete
Response Complete
Faces Request
Restore View
Apply Request Values
Process Events
Process Validations
Process Events
Render Response
Response Complete
Response Complete
Faces Response
Render Response
Process Events
Invoke Application
Process Events
Update Model Values
Conversion Errors / Render Response
Validation / Conversion Errors / Render Response



```
<h:inputText value="#{addressBook.filter}" id="filter">
    <f:ajax event="keyup" execute="filter" render="table"/>
</h:inputText>

<h:dataTable id="table" value="#{addressBook.names}" var="name">
    <h:column>#{name}</h:column>
</h:dataTable>
```

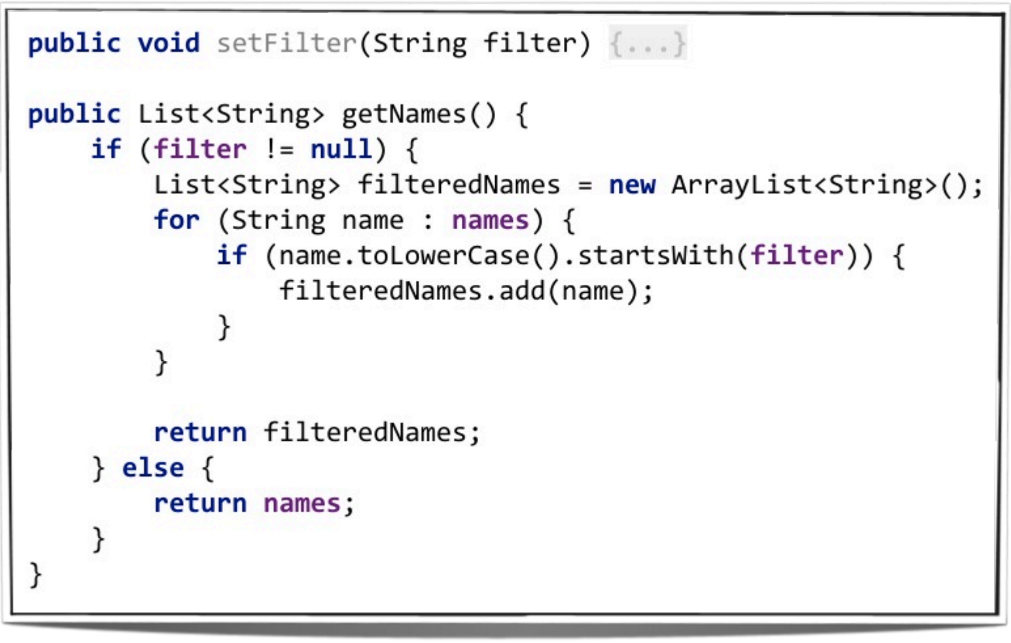




```
<h:inputText value="#{addressBook.filter}" id="filter">
    <f:ajax event="keyup" execute="filter" render="table"/>
</h:inputText>

<h:dataTable id="table" value="#{addressBook.names}" var="name">
    <h:column>#{name}</h:column>
</h:dataTable>
```

Managed bean

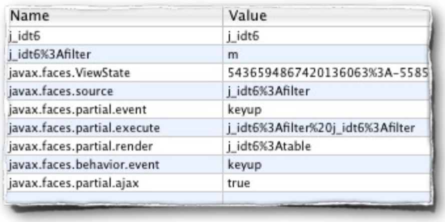
Nothing Ajax related in the managed bean



```
public void setFilter(String filter) {...}

public List<String> getNames() {
    if (filter != null) {
        List<String> filteredNames = new ArrayList<String>();
        for (String name : names) {
            if (name.toLowerCase().startsWith(filter)) {
                filteredNames.add(name);
            }
        }

        return filteredNames;
    } else {
        return names;
    }
}
```

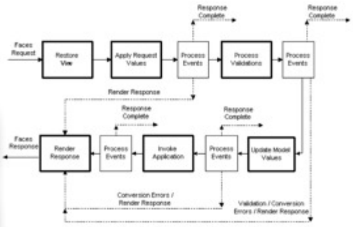
Ajax request

onKeyUp → XmlHttpRequest

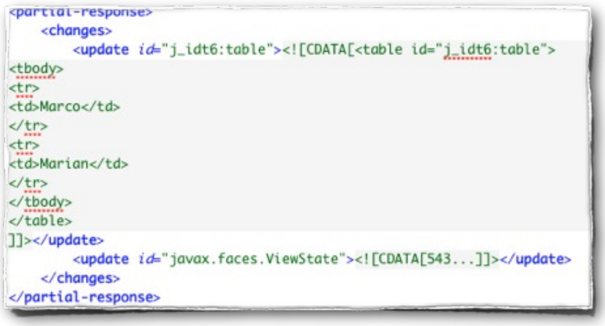


Name	Value
j_idt6	j_idt6
j_idt6%3Afilter	m
javax.faces.ViewState	5436594867420136063%3A-5585
javax.faces.source	j_idt6%3Afilter
javax.faces.partial.event	keyup
javax.faces.partial.execute	j_idt6%3Afilter%20j_idt6%3Afilter
javax.faces.partial.render	j_idt6%3Atable
javax.faces.behavior.event	keyup
javax.faces.partial.ajax	true

JSF

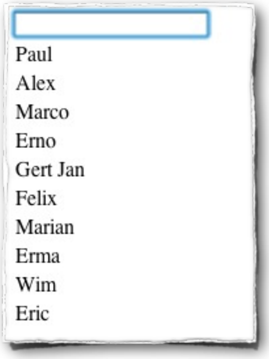


response



```
<partial-response>
    <changes>
        <update id="j_idt6:table"><![CDATA[<table id="j_idt6:table">
<tbody>
<tr>
<td>Marco</td>
</tr>
<tr>
<td>Marian</td>
</tr>
</tbody>
</table>
]]></update>
        <update id="javax.faces.ViewState"><![CDATA[543...]]></update>
    </changes>
</partial-response>
```

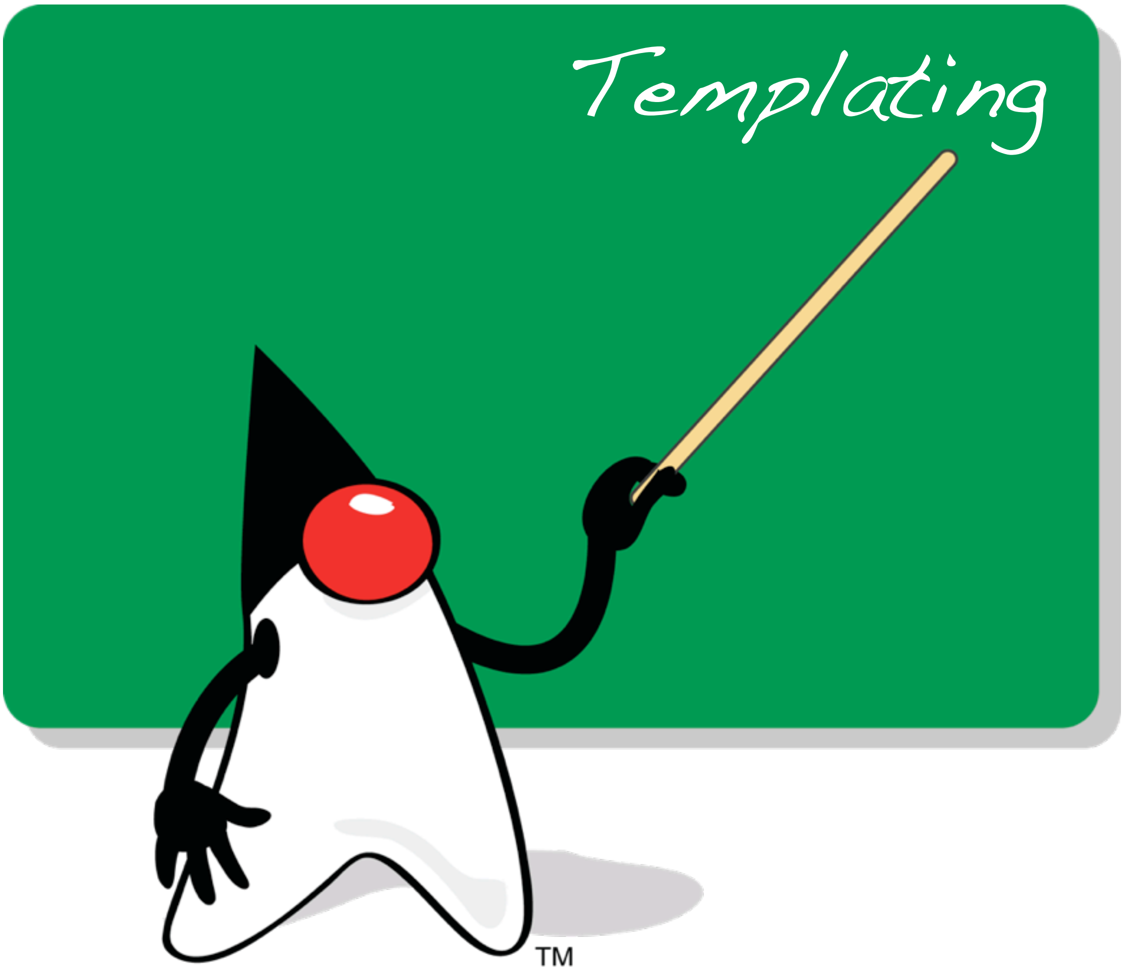
update



Paul
Alex
Marco
Erno
Gert Jan
Felix
Marian
Erma
Wim
Eric

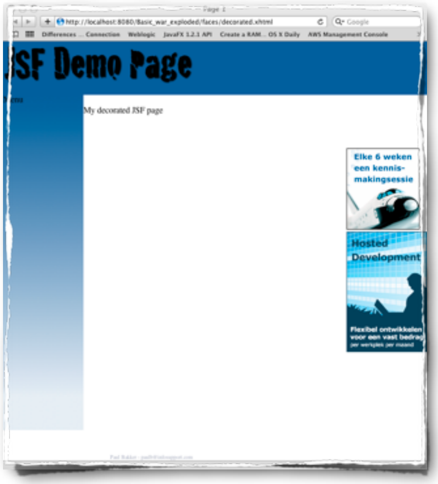
f:ajax properties

Property	Description
event	A String identifying the type of event the Ajax action will apply to.
execute	Identifiers of components that will participate in the "execute" portion of the Request Processing Lifecycle. Any of the keywords "@this", "@form", "@all", "@none" may be specified in the identifier list.
immediate	If "true" behavior events generated from this behavior are broadcast during Apply Request Values phase.
listener	Method expression referencing a method that will be called when an AjaxBehaviorEvent has been broadcast for the listener.
onevent	The name of the JavaScript function that will handle UI events.
oneerror	The name of the JavaScript function that will handle errors.
render	Evaluates to Collection. Identifiers of components that will participate in the "render" portion of the Request Processing Lifecycle. Any of the keywords "@this", "@form", "@all", "@none" may be specified in the identifier list.



Templating

- □ Create re-usable user interfaces
- □ A template decorates a page



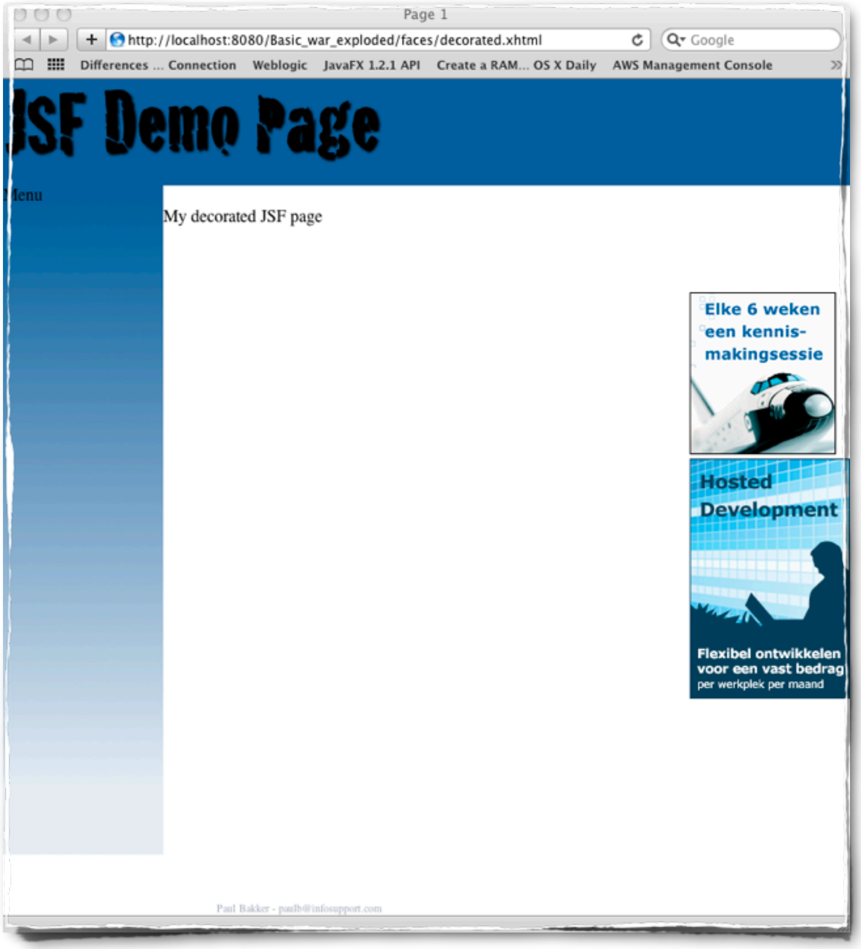
Page 1
http://localhost:8080/Basic_war_exploded/faces/decorated.xhtml
Google
Differences ... Connection Weblogic JavaFX 1.2.1 API Create a RAM... OS X Daily AWS Management Console

JSF Demo Page

Menu

My decorated JSF page

Elke 6 weken een kennis-makingssessie
Hosted Development
Flexibel ontwikkelen voor een vast bedrag per werkplek per maand


Page 1
http://localhost:8080/Basic_war_exploded/faces/decorated.xhtml
Google
Differences ... Connection
Weblogic
JavaFX 1.2.1 API
Create a RAM... OS X Daily
AWS Management Console

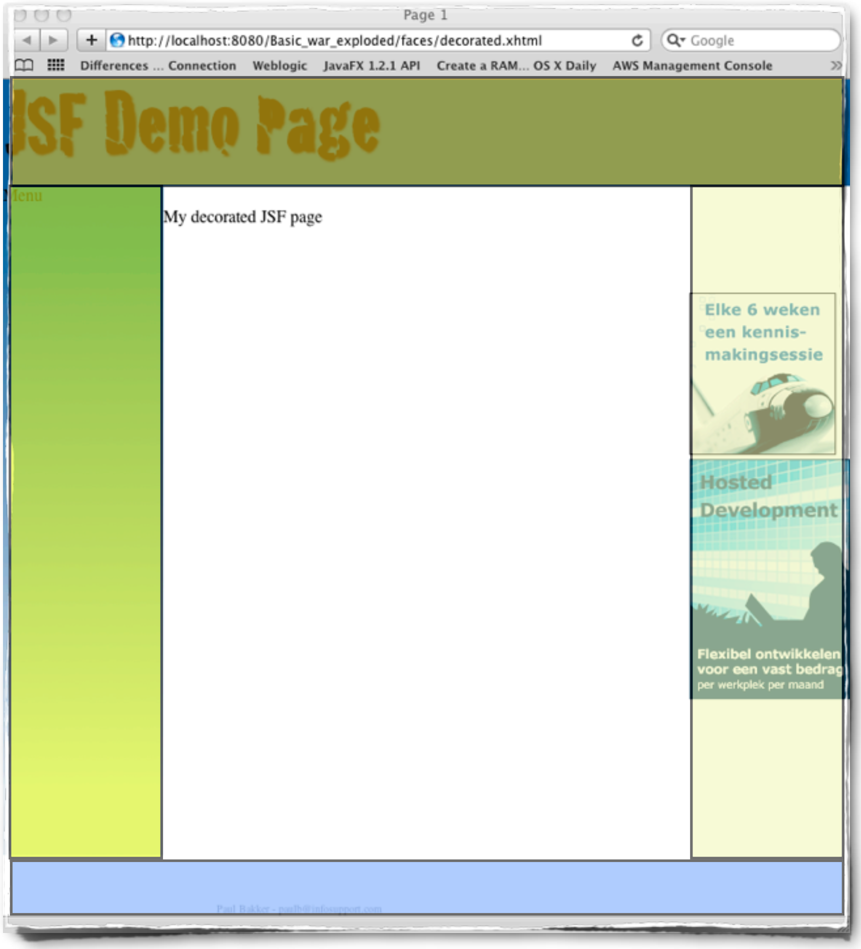
JSF Demo Page

Menu

My decorated JSF page

Elke 6 weken een kennis-makingsessie
Hosted Development
Flexibel ontwikkelen voor een vast bedrag per werkplek per maand

Paul Bakker - paulb@infosupport.com


Page 1
http://localhost:8080/Basic_war_exploded/faces/decorated.xhtml
Google
Differences ... Connection
Weblogic
JavaFX 1.2.1 API
Create a RAM... OS X Daily
AWS Management Console

JSF Demo Page

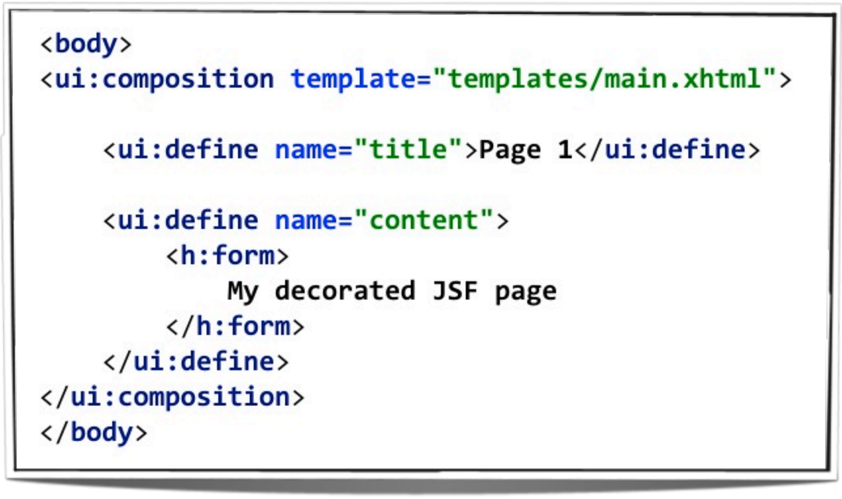
Menu

My decorated JSF page

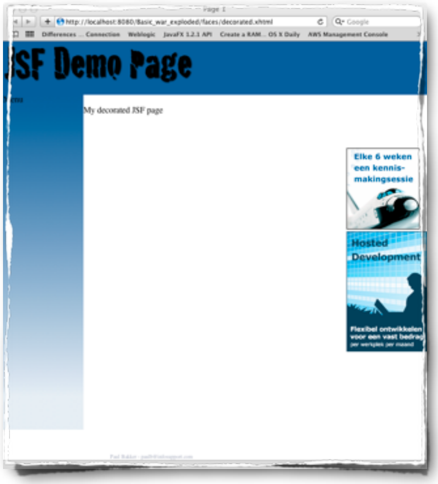
Elke 6 weken een kennis-makingsessie
Hosted Development
Flexibel ontwikkelen voor een vast bedrag per werkplek per maand

Paul Bakker - paulb@infosupport.com

decorated.xhtml



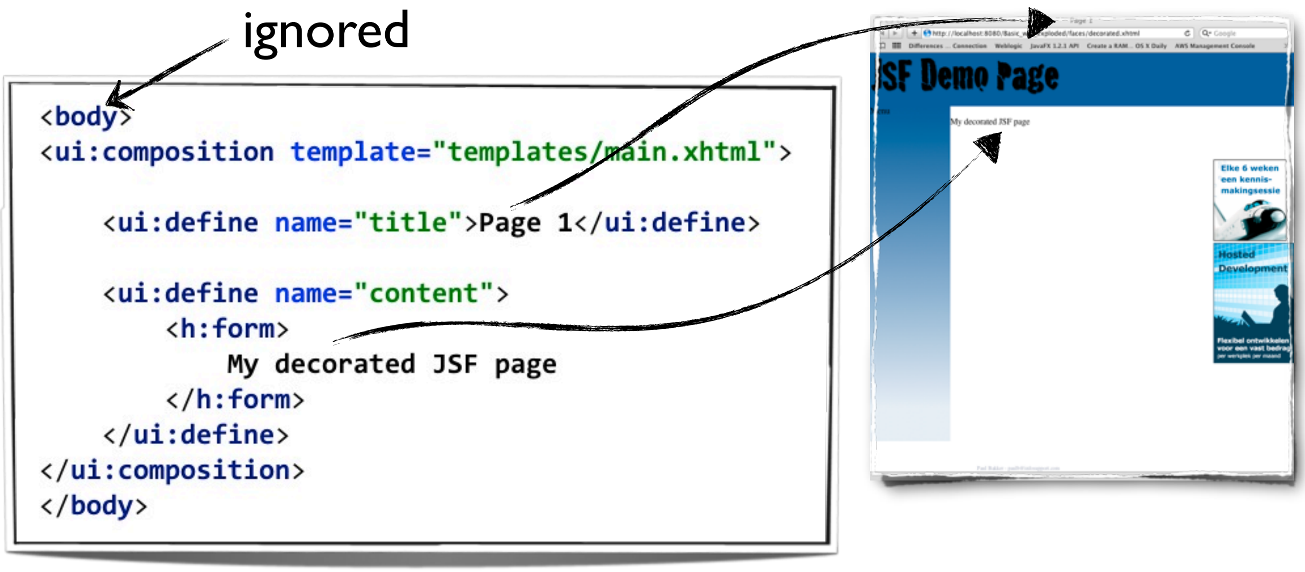
```
<body>
<ui:composition template="templates/main.xhtml">

    <ui:define name="title">Page 1</ui:define>

    <ui:define name="content">
        <h:form>
            My decorated JSF page
        </h:form>
    </ui:define>
</ui:composition>
</body>
```



Page 1
http://localhost:8080/Basic_war_exploded/faces/decorated.xhtml
Google
Differences ... Connection
Weblogic
JavaFX 1.2.1 API
Create a RAM... OS X Daily
AWS Management Console
JSF Demo Page
Menu
Elke 6 weken een kennis-makingsessie
Hosted Development
Flexibel ontwikkelen voor een vast bedrag per werkplek per maand

My decorated JSF page

decorated.xhtml

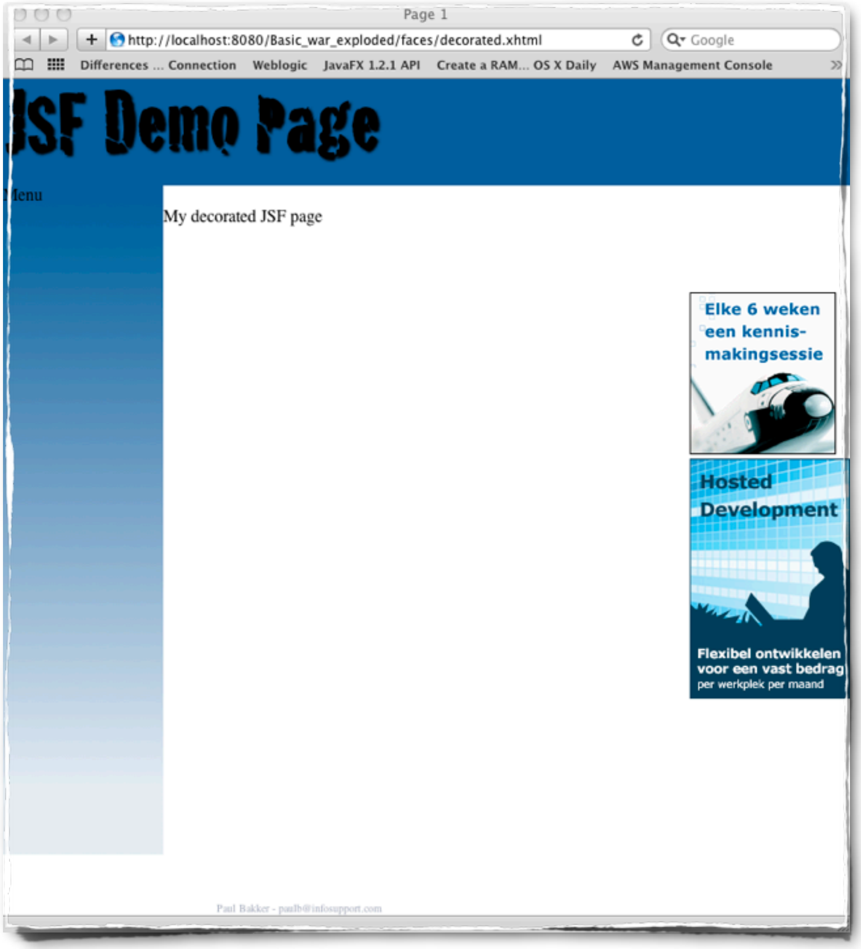


ignored

```
<body>
<ui:composition template="templates/main.xhtml">

    <ui:define name="title">Page 1</ui:define>

    <ui:define name="content">
        <h:form>
            My decorated JSF page
        </h:form>
    </ui:define>
</ui:composition>
</body>
```

Page 1
http://localhost:8080/Basic_war_exploded/faces/decorated.xhtml
JSF Demo Page
Elke 6 weken een kennis-makingsessie
Hosted Development
Flexibel ontwikkelen voor een vast bedrag per werkplek per maand

My decorated JSF page



Page 1
http://localhost:8080/Basic_war_exploded/faces/decorated.xhtml
Google
Differences ... Connection
Weblogic
JavaFX 1.2.1 API
Create a RAM... OS X Daily
AWS Management Console
Elke 6 weken een kennis- makingssessie
Hosted Development
Flexibel ontwikkelen voor een vast bedrag per werkplek per maand

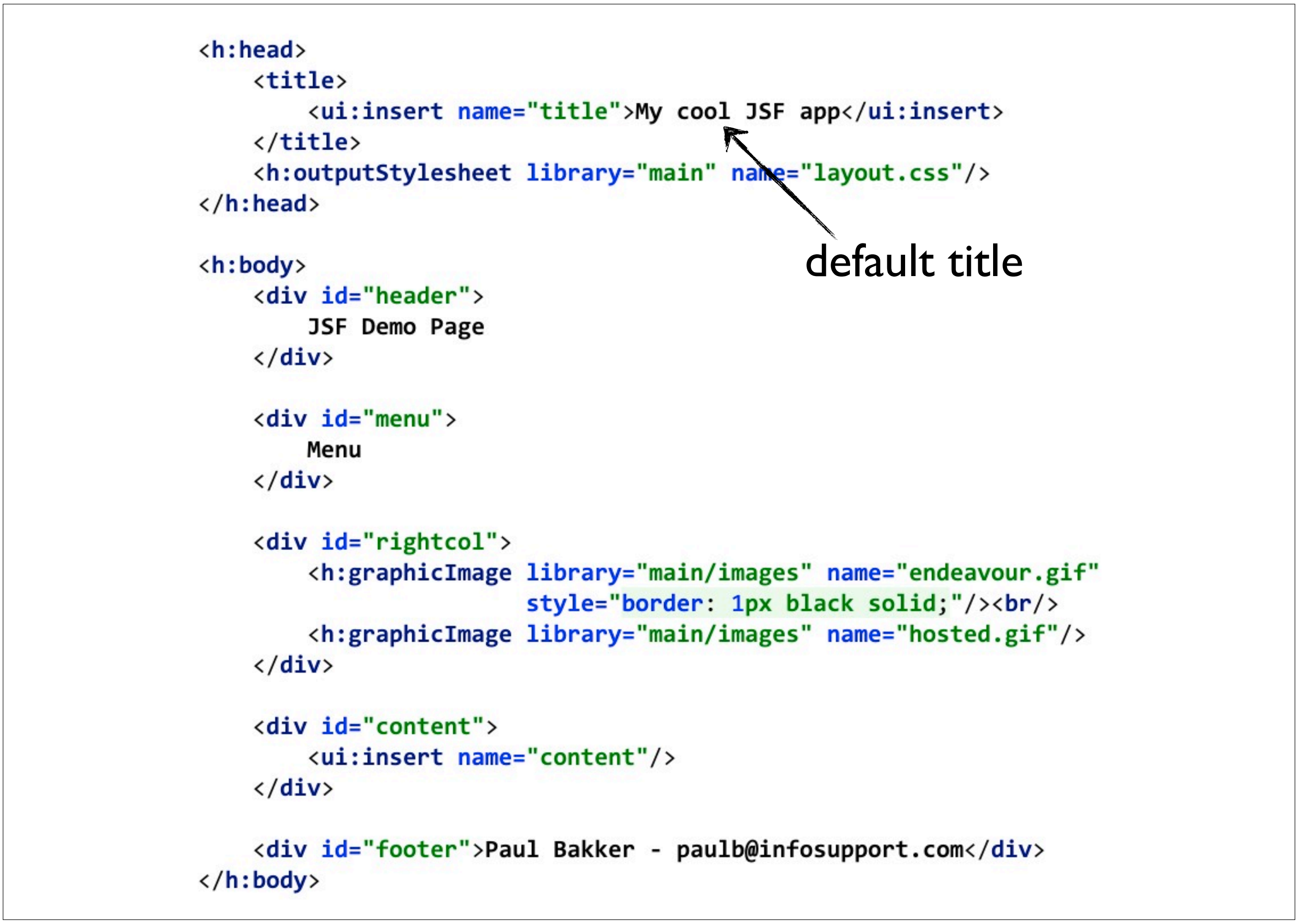
JSF Demo Page

Menu

My decorated JSF page

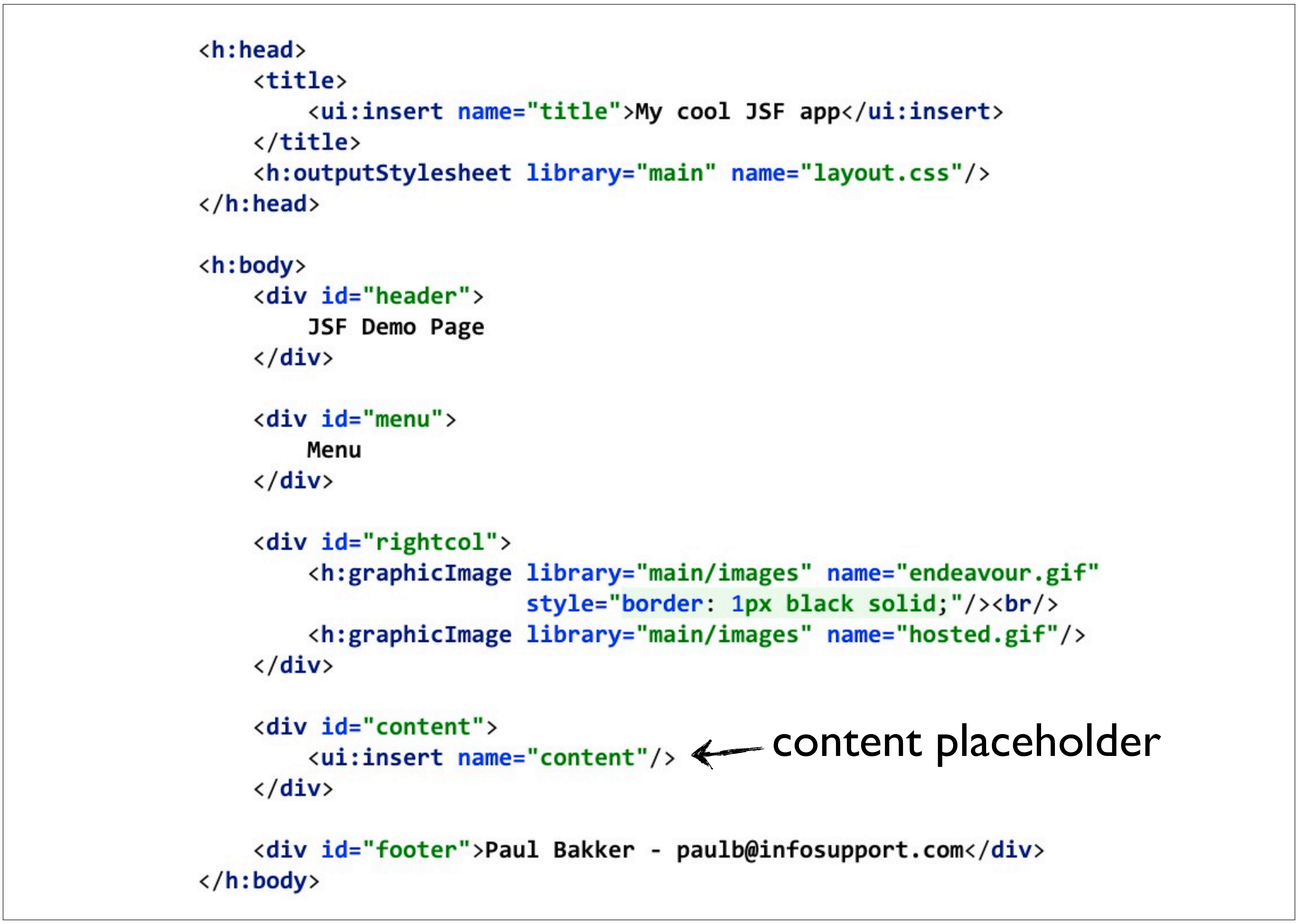
Paul Bakker - paulb@infosupport.com

```
<h:head>
    <title>
        <ui:insert name="title">My cool JSF app</ui:insert>
    </title>
    <h:outputStylesheet library="main" name="layout.css"/>
</h:head>

<h:body>
    <div id="header">
        JSF Demo Page
    </div>

    <div id="menu">
        Menu
    </div>

    <div id="rightcol">
        <h:graphicImage library="main/images" name="endeavour.gif"
                        style="border: 1px black solid;"/><br/>
        <h:graphicImage library="main/images" name="hosted.gif"/>
    </div>

    <div id="content">
        <ui:insert name="content"/>
    </div>

    <div id="footer">Paul Bakker - paulb@infosupport.com</div>
</h:body>
```



```
<h:head>
    <title>
        <ui:insert name="title">My cool JSF app</ui:insert>
    </title>
    <h:outputStylesheet library="main" name="layout.css"/>
</h:head>

<h:body>
    <div id="header">
        JSF Demo Page
    </div>

    <div id="menu">
        Menu
    </div>

    <div id="rightcol">
        <h:graphicImage library="main/images" name="endeavour.gif"
                        style="border: 1px black solid;"/><br/>
        <h:graphicImage library="main/images" name="hosted.gif"/>
    </div>

    <div id="content">
        <ui:insert name="content"/>
    </div>

    <div id="footer">Paul Bakker - paulb@infosupport.com</div>
</h:body>
```

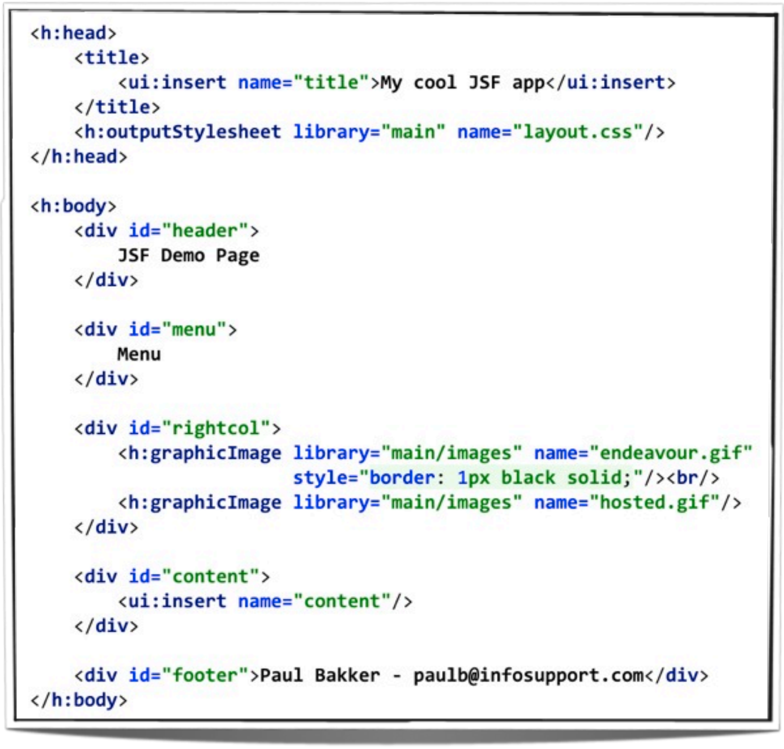
default title



```
<h:head>
    <title>
        <ui:insert name="title">My cool JSF app</ui:insert>
    </title>
    <h:outputStylesheet library="main" name="layout.css"/>
</h:head>

<h:body>
    <div id="header">
        JSF Demo Page
    </div>

    <div id="menu">
        Menu
    </div>

    <div id="rightcol">
        <h:graphicImage library="main/images" name="endeavour.gif"
                        style="border: 1px black solid;"/><br/>
        <h:graphicImage library="main/images" name="hosted.gif"/>
    </div>

    <div id="content">
        <ui:insert name="content"/>
    </div>

    <div id="footer">Paul Bakker - paulb@infosupport.com</div>
</h:body>
```

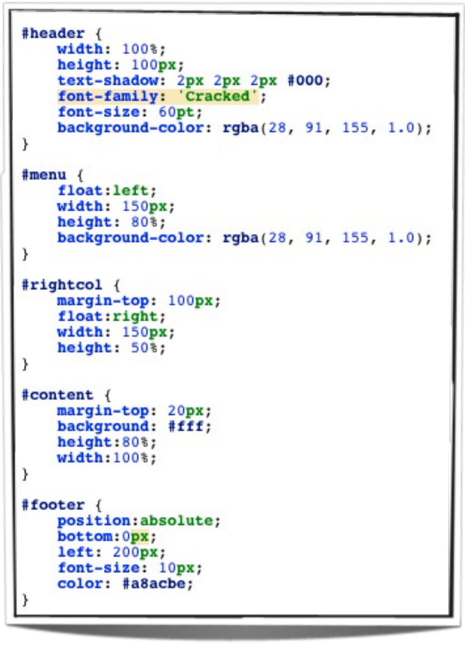
content placeholder

templates/main.xhtml



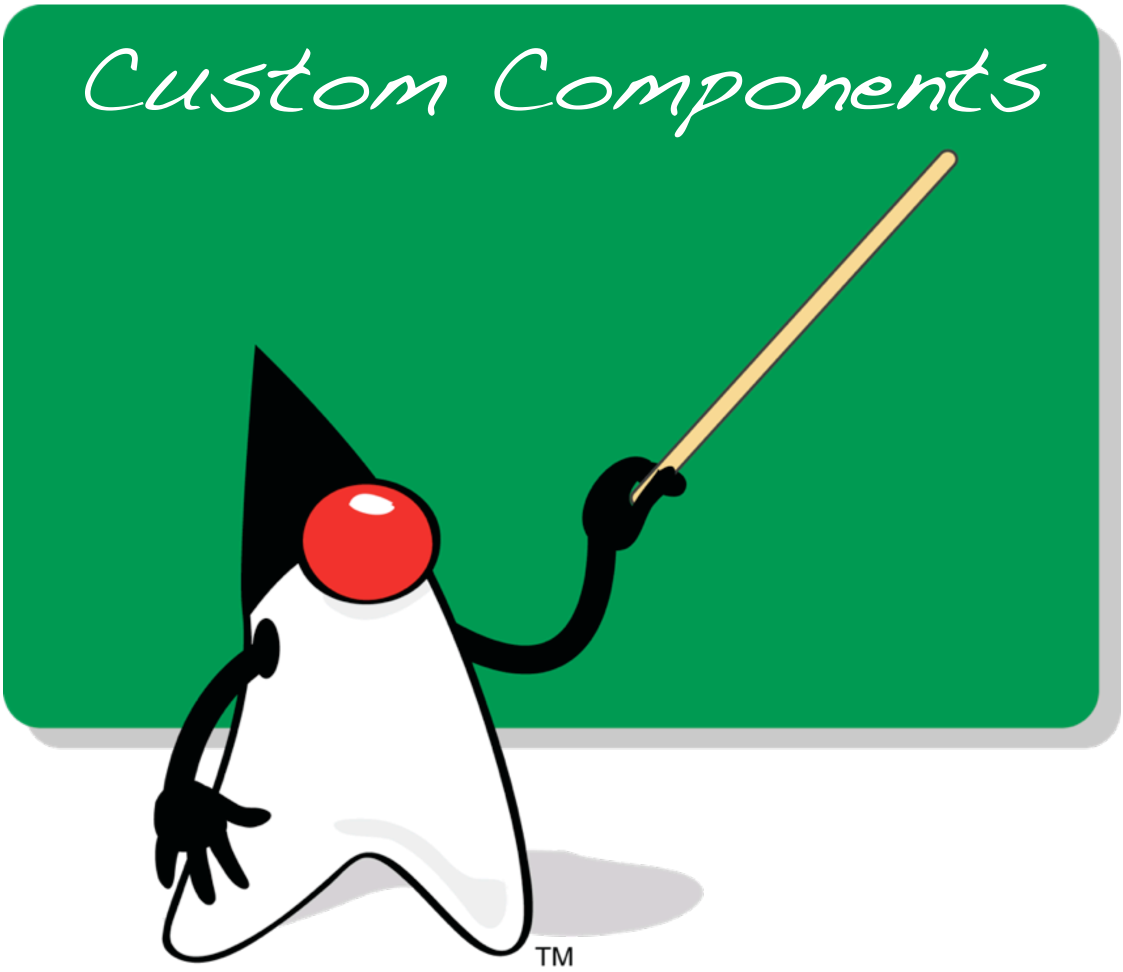
```
<h:head>
    <title>
        <ui:insert name="title">My cool JSF app</ui:insert>
    </title>
    <h:outputStylesheet library="main" name="layout.css"/>
</h:head>

<h:body>
    <div id="header">
        JSF Demo Page
    </div>

    <div id="menu">
        Menu
    </div>

    <div id="rightcol">
        <h:graphicImage library="main/images" name="endeavour.gif"
                        style="border: 1px black solid;"/><br/>
        <h:graphicImage library="main/images" name="hosted.gif"/>
    </div>

    <div id="content">
        <ui:insert name="content"/>
    </div>

    <div id="footer">Paul Bakker - paulb@infosupport.com</div>
</h:body>
```

layout.css



```
#header {
    width: 100%;
    height: 100px;
    text-shadow: 2px 2px 2px #000;
    font-family: 'Cracked';
    font-size: 60pt;
    background-color: rgba(28, 91, 155, 1.0);
}

#menu {
    float:left;
    width: 150px;
    height: 80%;
    background-color: rgba(28, 91, 155, 1.0);
}

#rightcol {
    margin-top: 100px;
    float:right;
    width: 150px;
    height: 50%;
}

#content {
    margin-top: 20px;
    background: #fff;
    height:80%;
    width:100%;
}

#footer {
    position:absolute;
    bottom:0px;
    left: 200px;
    font-size: 10px;
    color: #a8acbe;
}
```

Custom Components

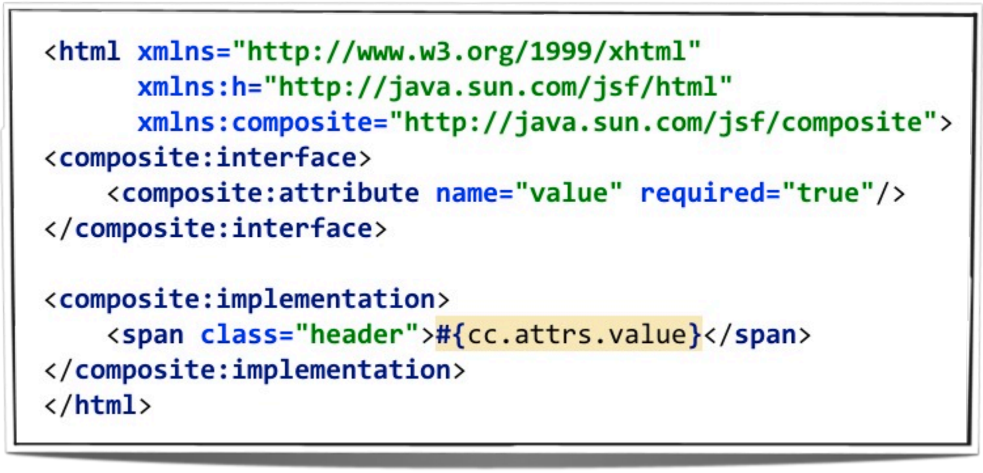


Composite components

- UI only components
- Define component in:
  - resources/[component lib]/[component].xhtml
- component:interface
  - Define component attributes
- component:implementation
  - Define component

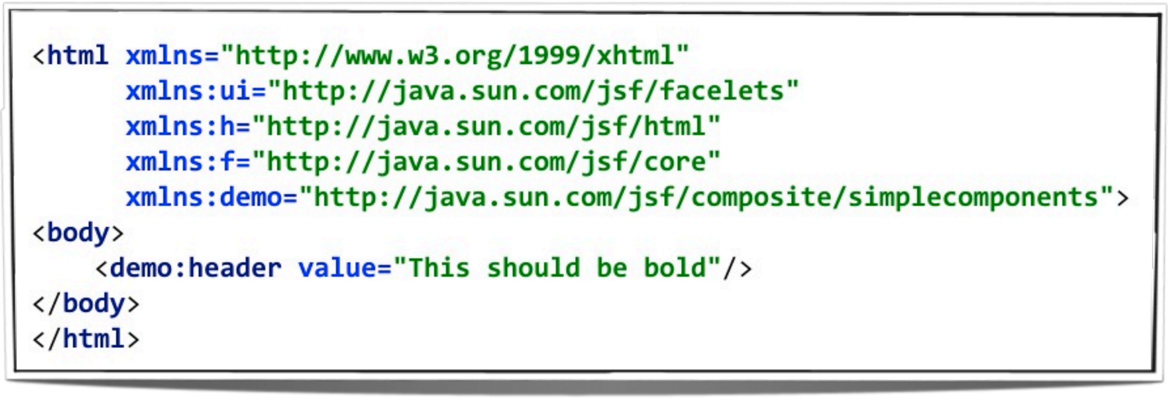
Defining a component

resources/simplecomponents/header.xhtml



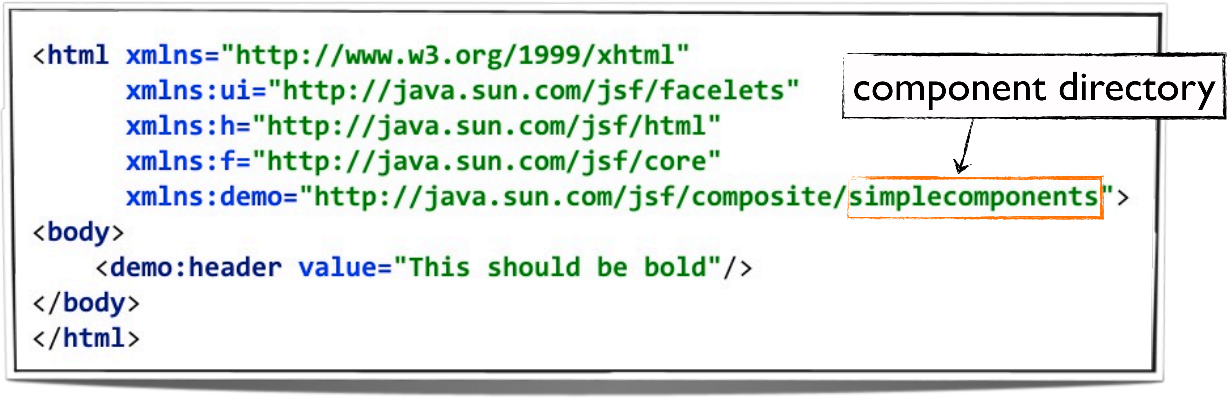
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:composite="http://java.sun.com/jsf/composite">
<composite:interface>
    <composite:attribute name="value" required="true"/>
</composite:interface>

<composite:implementation>
    <span class="header">#{cc.attrs.value}</span>
</composite:implementation>
</html>
```

Using a component



```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:demo="http://java.sun.com/jsf/composite/simplecomponents">
<body>
    <demo:header value="This should be bold"/>
</body>
</html>
```

Using a component



```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:demo="http://java.sun.com/jsf/composite/simplecomponents">
<body>
    <demo:header value="This should be bold"/>
</body>
</html>
```

component directory