Introduction to **J**ava **S**erver **F**aces
(covered topics)

- Reasons to use „Java Server Faces“
- JSF application structure
- JSF component model
- JSF managed beans model
- JSF navigation model
- JSF error validation model
- JSF versions (1.0, 1.1, 1.2, 2.0)
- JSF implementations

Some reasons to use JSF

- JSF is standard for Java web framework.
- JSF is MVC model 2 based framework => separation between UI, business logic and data.
- JSF is component oriented and event driven framework.
- Choice between many JSF implementations, both free and commercial.
- JSF vendors provide rich set third party UI components, both free and commercial.
- IDE support (Eclispe, NetBeans, JBuilder, ...).

Common WAR structure

CommonWebApplication.war
- index.jsp
- another.jsp
- **WEB-INF**
  - web.xml
  - **lib**
    - somelibrary.jar
    - anotherlibrary.jar
  - **classes**
    - package
      - structure
        - SomeClass.class
        - AnotherClass.class

JSF enabled WAR structure (1 of 2)

WebApplicationWithJSF.war
- index.jsp
- another.jsp
- **WEB-INF**
  - web.xml *(with changes)*
  - *faces-config.xml*
  - **lib**
    - *jsflibrary.jar*
    - anotherlibrary.jar
  - **classes**
    - package
      - structure
        - SomeClass.class
        - AnotherClass.class

JSF enabled WAR structure (2 of 2)

- web.xml – includes JSF specific configuration (JSF servlet mappings, additional configuration files, specific JSF parameters, ...).
- faces-config.xml – this is the default JSF configuration file. It includes navigation rules, configuration of backing beans, converters, validators...
- jsflibrary.jar – one or more JAR files which represents the JSF implementation.

Quick overview of JSF components (1 of 2)

- JSF components are custom tags.
- JSF standard defines two types of JSF component tags: html tags and core tags.
- HTML tags represent html components like text fields, buttons, form, ... There are 25 standard html tags.
- Core tags allows you to take advantages of features of JSF framework, like validation, conversion, event handling, ... There are 18 standard core tags.

Quick overview of JSF components (2 of 2)

- JSF input components can be restricted with validators and type converters.
- JSF command components can raise events on the server (JSF is event driven framework).
- JSF components use custom implementation of EL (expression language) to access data in managed beans.

JSF components (examples 1 of 3)

JSF tag in JSP page:

```
<h:inputText id="someId" value="edit me" />
```

Rendered HTML response may be like this:

```
<input type ="text" id="someId" value="edit me" />
```

JSF components (examples 2 of 3)

JSF tag in JSP page:

```
<h:commandButton id="someId" value="Click to edit"
action="goEdit" />
```

Rendered HTML response may be like this:

```
<input type ="submit" id="someId" value="Click to edit" />
```

JSF components (examples 3 of 3)

Content of JSF enabled JSP page:

```
<%@ taglib uri="http://java.sun.com/jsf/core prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html prefix="h" %>
...HTML tags
<f:view>
  <h:form id="someId">
  ...JSF component tags
  </h:form>
<f:view>
...HTML tags
```

JSF Life Cycle (propaganda)

- JSF life cycle (multiple phases) is conceptually different from JSP life cycle (single phase with direct evaluation).
- JSF life cycle phases are not compatible with JSP life cycle => JSP and JSF technologies experience difficulties with shared resources (accessing one and the same data via EL for example) => software developers experience severe difficulties too.

JSF Life Cycle Phases (1 of 7)

Phase 1: Restore view phase

Phase 2: Apply request values phase

Phase 3: Process validations phase

Phase 4: Update model values phase

Phase 5: Invoke application phase

Phase 6: Render response phase

JSF Life Cycle Phases (2 of 7)

Phase 1: **Restore view**

- Invoked by clicking a link, button, etc.
- View is created or restored (in case of POST request) and then saved in FacesContext.
- Binding of components to event handlers and validators.
- Component tree is generated or rebuilt (in case of POST request).

JSF Life Cycle Phases (3 of 7)

Phase 2: **Apply request values**

- Components retrieve their new values and store them locally.
- Component values are converted.
- Error messages are queued in FacesContext.
- Immediate – conversion, validation, bean updates and events are processed in this phase.

JSF Life Cycle Phases (4 of 7)

Phase 3: **Process validations**

- Local component values are validated using the validation rules registered for the component.
- Validation error messages and conversion error messages are added to FacesContext.
- If validation fails – proceed to Render response phase.

JSF Life Cycle Phases (5 of 7)

Phase 4: **Update model values**

- Local component values are set to the corresponding managed bean properties.
- If local component value type is incompatible with managed bean property type – proceed to Render response phase.

JSF Life Cycle Phases (6 of 7)

Phase 5: **Invoke application**

- Server side events are invoked.
- Application level events are applied (for example navigation rules).

JSF Life Cycle Phases (7 of 7)

Phase 6: **Render response**

- On initial request components are added to the component tree.
- Content view state is saved after the response.

JSF Life Cycle Phases (overview 1 of 2)

- Restore view - creates new page or restores the previous page.
- Apply request values - set component submitted values to request values.
- Process validations - Validate component values. Set component values to submitted values if valid.

JSF Life Cycle Phases (overview 2 of 2)

- Update model values - set backing bean values to component values.
- Invoke application - execute actionListeners and actions.
- Render response - return response and save the view state.

How the „immediate“ atribute works

- JSF input and command components have special attribute, called immediate.
- If immediate attribute is set to „true“– validation, conversion and events are processed in Apply request values phase.

Why is this attribute so special?

- Immediate attribute allows to navigate from one page to another while ignoring validation.
- We can make one or more components „high priority“ for validation. This can reduce number of error messages shown.

Immediate attribute examples (1 of 2)

JSF tag in JSP page:

```
<h:inputText id="someId" value="edit me"
  immediate="true" required="true" />
```

Result:

- This component will be validated in Apply request values phase.
- If the value is empty – error message will appear.

Other non-immediate component error messages are ignored.

Immediate attribute examples (2 of 2)

JSF tag in JSP page:

```
<h:commandButton id="someId" value="Click to go back"
 action="goBack" immediate="true" />
```

Result:

- The action method is invoked in Apply request values phase.
- The action method is invoked before any non-imediate value validation and before any managed bean updates.

JSF Managed Beans

- Managed beans (for non-command components) are Java classes.
- Managed beans represent the Model part of MVC in JSF (for non-command components).
- Managed beans can have „Controller“ functionality (for command components).
- Managed beans are described and configured in faces-config.xml.
- Managed beans can have different scopes (request, session, aplication).

Simple managed bean example (1 of 3)

Managed bean class MyBean.java:

```
package sample;

public class MyBean {
  private String myName;

  public MyBean() {
    myName = "Ivan Davidov";
  }

  public void setMyName(String newName) {
    myName = newName;
  }

  public String getMyName() {
    return myName;
  }
}
```

Simple managed bean example (2 of 3)

XML snippet in faces-config.xml:

```
<managed-bean>
 <managed-bean-name>myBean</managed-bean-name>
 <managed-bean-class>sample.MyBean</managed-bean-class>
 <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Simple managed bean example (3 of 3)

How to use in JSF components:

```
<%@ taglib uri="http://java.sun.com/jsf/core prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html prefix="h" %>
...HTML tags
<f:view>
  <h:form id="someId">
    <h:inputText id="someId" value="#{myBean.myName}" />
  </h:form>
<f:view>
...HTML tags
```

JSF Navigation Model

- Navigation model defines rules that allow you to define navigation from view to view.
- Navigations are defined in faces-config.xml.

Simple navigation example (1 of 3)

faces-config.xml snippet for navigation:

```
<navigation-rule>
  <from-view-id>/signin/login.jsp</from-view-id>
  <navigation-case>
    <from-outcome>correctLogin</from-outcome>
    <to-view-id>/signin/success.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>incorrectLogin</from-outcome>
    <to-view-id>/signin/wrongpass.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Simple navigation example (2 of 3)

login.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html prefix="h" %>
...HTML tags
<f:view>
  <h:form id="someId">
    <h:inputText id="user" value="#{signinBean.user}" />
    <h:inputSecret id="pass" value="#{signinBean.pass}" />
    <h:commandButton id="pass" value="Log in"
       action="#{signinBean.loginMethod}" />
  </h:form>
<f:view>
...HTML tags
```

Simple navigation example (3 of 3)

SigninBean.java

```
package sample;

  public class SigninBean {
    // Standard Java bean properties goes here

    public String loginMethod() {
      if (getUser().equals("ivan") &&
          getPass().equals("12345")) {
        // login is correct
        return "correctLogin";
      }

      // login is incorrect
      return "incorrectLogin";
    }
  }
```

JSF Error Validation Model

There are four forms of validation within JSF:

- Built-in validation components.
- Application level validation.
- Custom validation components.
- Validation methods in backing beans.

Built-in validation components

Core tag validators have common signature:

```
<f:validateXxx ...attributes goes here... />
```

Examples:

- `<f:validateLength minimum="5" maximum="15" />`
- `<f:validateLongRange minimum="2" maximum="18" />`
- `<f:validateDoubleRange minimum="2.4" maximum="7.18" />`

Simple validation example

register.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html prefix="h" %>
...HTML tags
<f:view>
  <h:form id="regForm">
  ...JSF tags
    <h:messages showSummary="true" showDetail="false"
       layout="table" />
    <h:inputText id="ageId" value="#{regBean.userAge}">
      <f:validateLongRange minimum="0" maximum="150" />
    </h:inputText>
    <h:message id="msgAge" for="ageId" />
  ...More JSF tags
  </h:form>
<f:view>
...HTML tags
```

Application level validation

Implemented as backend code inside the action method of a managed bean.

Advantages and disadvantages:

- Easy to implement.
- No nead for separate class (custom validator).
- Occurs after all other forms of validation.
- Difficult to manage in large applications.

Application level validation (example)

Part of MyBean.java

```
public String loginMethod() {
  FacesContext ctx = FacesContext.getCurrentInstance();
  if (getUser() == null || getUser().equals("") {
    // user field is empty
    FacesMessage message = new FacesMessage();
    message.setSeverity(FacesMessage.SEVERITY_ERROR);
    message.setSummary("User field is empty.");
    // add error message to context
    ctx.addMessage("regForm:ageId", message);

    return "";
  }

  ...other backend application logic (don't forget to
return String at the end).
}
```

Validation methods in managed beans

- Implemented as method in managed bean with the following signature:

```
public void methodName(FacesContext context,
                       UIComponent component,
                       Object value);
```

Validation method (example 1 of 2)

Part of MyBean.java

```
public void validateUser(FacesContext context,
                         UIComponent component,
                         Object value) {

  String userName = (String) value;

  if (userName == null || userName.equals("")) {
    // validation fails for this component.
    ((UIInput)component).setValid(false);
    FacesMessage message = new FacesMessage("User field
                                             is empty");

    context.addMessage(component.getClientId(context),
                       message);
  }
}
```

Validation method (example 1 of 2)

register.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html prefix="h" %>
...HTML tags
<f:view>
  <h:form id="regForm">
  ...JSF tags
    <h:messages showSummary="true" showDetail="false"
       layout="table" />
    <h:inputText id="userId" value="#{regBean.user}"
     validator="#{regBean.validateUser}" required="true" />
    <h:message id="msgUser" for="userId" />
  ...More JSF tags
  </h:form>
<f:view>
...HTML tags
```

JSF Versions

- JSF 1.0 – the first JSF standard.
- JSF 1.1 – bugfix release, no changes.
- JSF 1.2 – added new features (unified EL, better error messages support, etc.).
- JSF 2.0 – future release, expected in 2008. Expected better error handling model, skin support, etc.

JSF Implementations

- Sun Reference Implementation (Sun RI).
- Apache MyFaces.

Additional resources

- JSF tutorials: http://jsftutorials.net
- JSF introduction step by step: http://www.roseindia.net/jsf/