JavaServer Faces

2.2

Anders Møller and Mathias Schwarz <schwarz@cs.au.dk>

Overview

- **Motivation**
- Hello World
- Requests and responses
- Data storage
- Navigation
- Guessing Game in JSF
- Advanced features
- Deployment

Motivation: Number Guessing Game

- Guess a number between 1 and 100
- High-score list
- State at different levels
  - transient: the most recent guess
  - session: secret number and number of guesses
  - shared: high-score list
- Non-trivial control-flow

Servlet Implementation (1/4)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Random;

public class Play extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws IOException, ServletException {
    HttpSession session = request.getSession();
    ServletContext servletcontext = getServletContext();
    String submit = request.getParameter("submit");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>The Number Guessing Game</title></head>");
    out.println("<h3>The Number Guessing Game</h3>");
```

- dataflow
- application flow
- response

Servlet Implementation (2/4)

```
if (submit==null) {
  session.setAttribute("guesses",0);
  session.setAttribute("secret",new Random().nextInt(100)+1);
  out.println("<form method=post action=Play>");
  out.println("Guess a number between 1 and 100");
  out.println("<input type=text name=number>");
  out.println("<input type=submit name=submit value=Guess>");
  out.println("</form>");
} else if (submit.equals("Register")) {
  servletcontext.setAttribute("hiscore",session.getAttribute("guesses"));
  servletcontext.setAttribute("hiname",request.getParameter("name"));
  out.println("Thanks for playing.");
} else {
  int number = Integer.parseInt(request.getParameter("number"));
  int secret = (Integer)session.getAttribute("secret");
  int guesses = (Integer)session.getAttribute("guesses")+1;
  session.setAttribute("guesses",guesses);
```

Servlet Implementation (3/4)

```
if (number != secret) {
  out.println("<form method=post action=Play>");
  if (number < secret)
    out.println("Your guess is too small, guess again.");
  else
    out.println("Your guess is too large, guess again.");
  out.println("<input type=text name=number>");
  out.println("<input type=submit name=submit value=Guess>");
  out.println("</form>");
} else {
  Integer hiscore = (Integer)servletcontext.getAttribute("hiscore");
  if (hiscore==null || guesses < hiscore) {
     out.println("<form method=post action=Play>");
     out.println("You have the new highscore of "+guesses+" guesses!<p>");
     out.println("Please enter your name:");
     out.println("<input type=text name=name>");
     out.println("<input type=submit name=submit value=Register>");
     out.println("</form>");
```

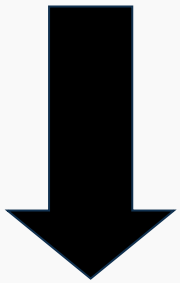
Servlet Implementation (4/4)

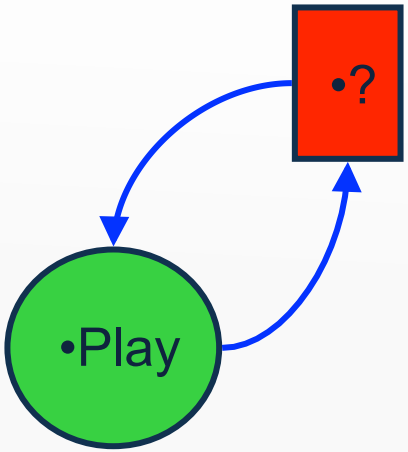
```
        } else {
          String hiname = (String)servletcontext.getAttribute("hiname");
          out.println("You used "+guesses+" guesses.<p>");
          out.println("The highscore is "+hiname+" with "+hiscore+" guesses.");
        }
      }
    }
    out.println("</body></html>");
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws IOException, ServletException {
    doPost(request,response);
  }
}
```

You're Doing It Wrong

- Obscure application flow
- Obscure generation of HTML pages
- No separation of concerns
- Difficult to maintain



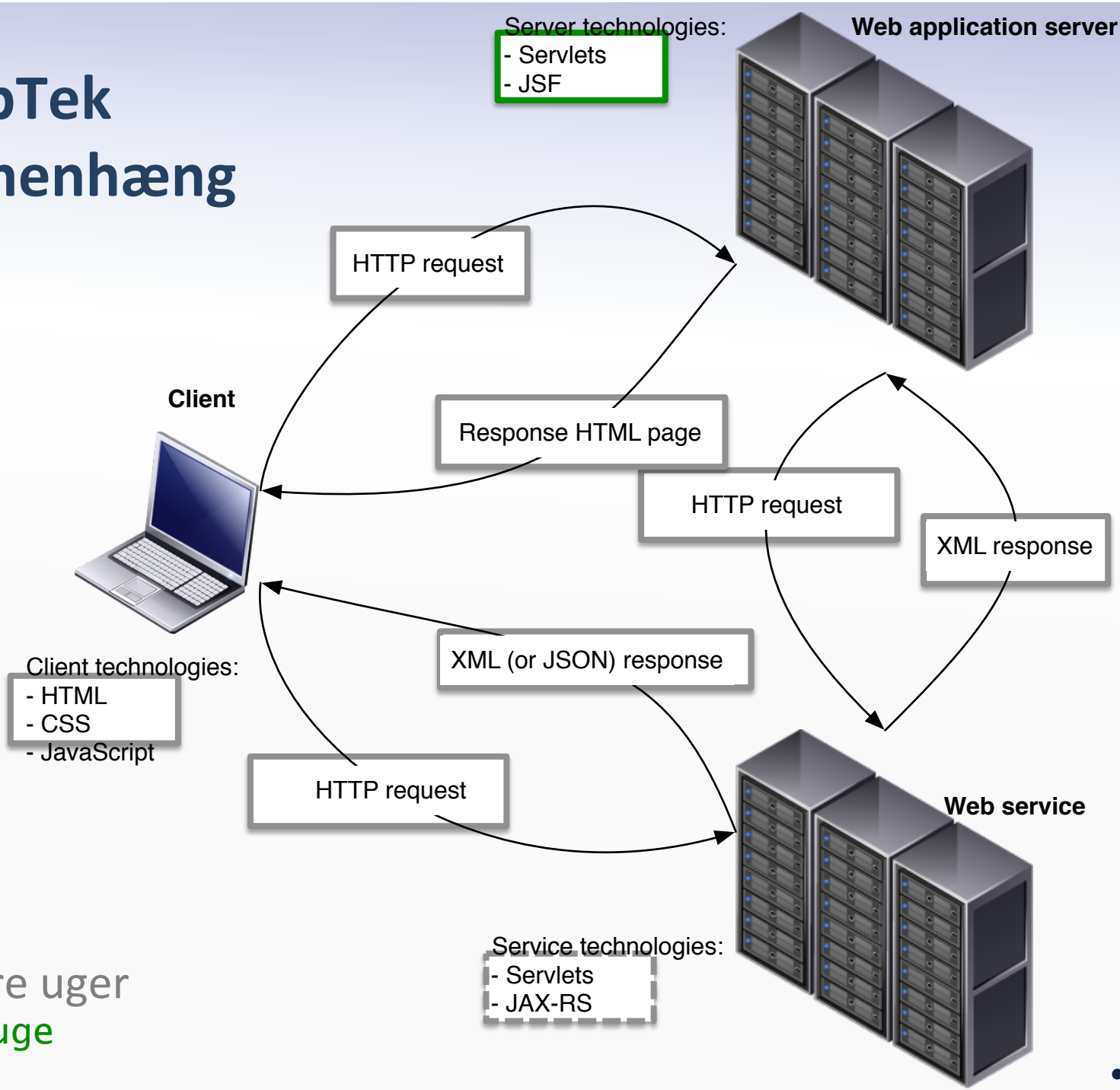


- Use the Model-View-Controller pattern

Motivation

- Java Servlets are a good foundation, but missing
  - HTML templates
    - We want: Reuseable, structured, without need to escape chars
  - Application code structure
    - We want: HTML, Java code clearly separated
  - Application flow structure
    - Which Servlets should be visited in which order by the client?
  - Input validation
    - We want: our code to not crash if "aa" is given as int input
- We need something on top of Servlets to handle this!

dWebTek sammenhæng

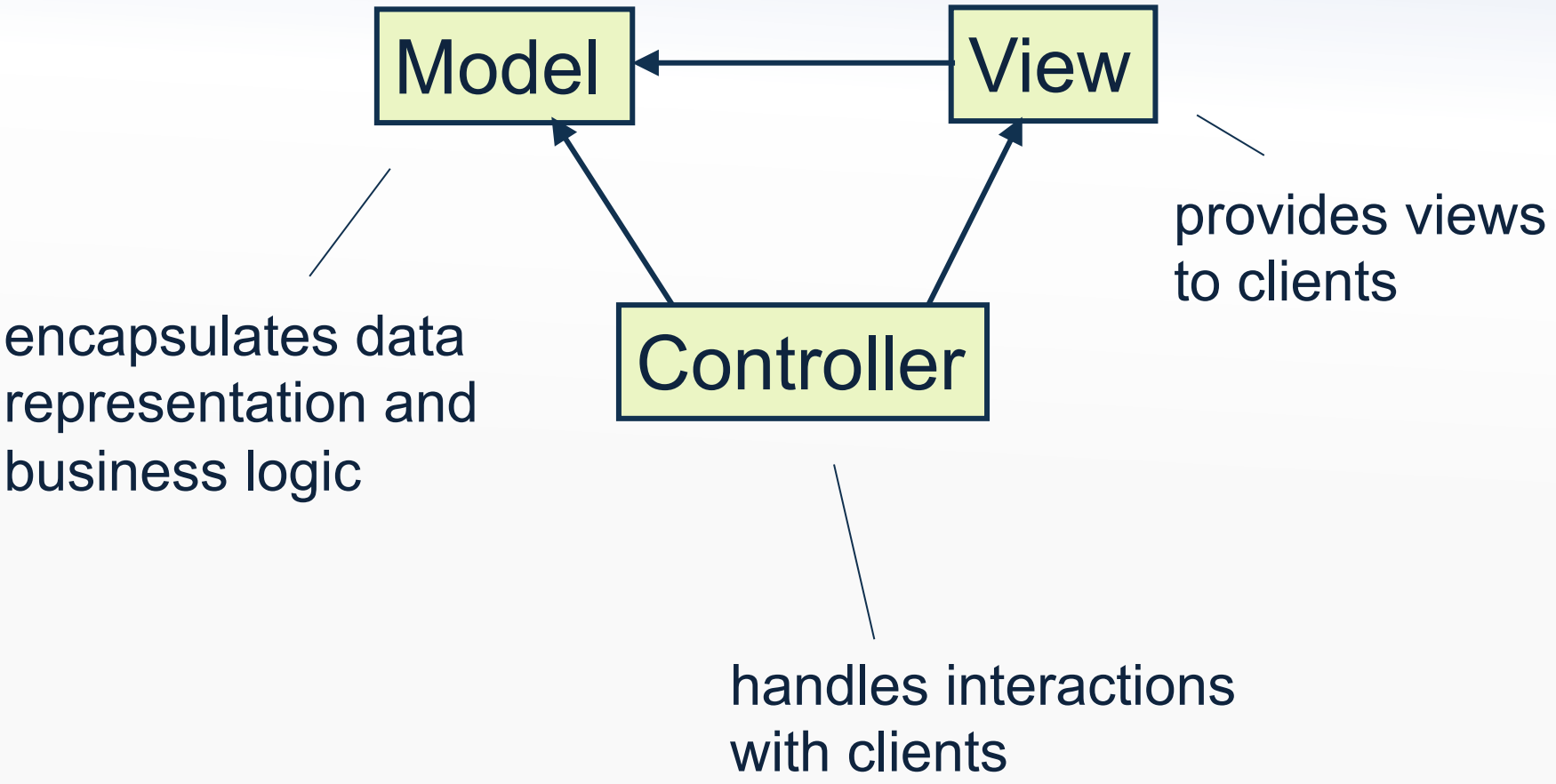


- Tidligere uger
- Denne uge

Historical background

- 1999: JSP 1.0
  - Simple template system on top of the servlet framework
  - Templates looked like XML but weren't
  - Built on compilation JSP -> servlets
  - Except for syntax, much like servlets in programming style
- 2004: JSF 1.0
  - Added navigation, state management to JSP
  - JSP still used as template language
  - Numerous stability and JSP integration problems
- 2009: JSF 2.0
  - JSP-less JSF with a proper XML template language

The Model-View-Controller Pattern



Main ideas

- MVC
  - Model: beans
    - Plain Java objects with setters, getters
  - View: JSF pages (using XML and Namespaces)
    - Facelets
    - Components
  - Controller: the FacesServlet

- v2.x is very different from 1.x (which used JSP)
- http://www.javaserverfaces.org

Overview

- Motivation
- **Hello World**
- Requests and responses
- Data storage
- Navigation
- Guessing Game in JSF
- Advanced features
- Deployment

Hello World

welcome.xhtml:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <title>Welcome!</title>
    </head>
    <body>
        <h2>What is your name?</h2>
        <h:form>
            <h:inputText value="#{helloBean.name}"/>
            <h:commandButton action="response" value="Submit"/>
        </h:form>
    </body>
</html>
```

Hello World

HelloBean.java:

```
package swt;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import java.io.Serializable;

@ManagedBean
@SessionScoped
public class HelloBean implements Serializable {

   private String name;

   public String getName() { return name; }

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

Hello World

response.xhtml:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <head><title>Response</title></head>
  <body>
    <h:form>
      <h2>Hello, #{helloBean.name}</h2>
      <h:commandButton value="Back" action="welcome" />
    </h:form>
  </body>
</html>
```

faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
       http://java.sun.com/xml/ns/javaee/web-facesconfig_2_1.xsd"
     version="2.1">

  <!- not required in this simple example -->

</faces-config>
```

Overview

- Motivation
- Hello World
- **Requests and responses**
- Data storage
- Navigation
- Guessing Game in JSF
- Advanced features
- Deployment

Requests

- Indirectly represented:
  - Data is *injected* into Java beans
    - Essentially a Java class with getters and setters
    - Parameter *foo* is set by calling *setFoo* etc
  - Values available by calling getters on beans (as we saw)
- Underlying framework is still Servlets
  - Hidden from the programmer

Request processing flow

- The URL `http://localhost:8080/MyApp/foo.jsf`
  1. is handled by JSF (the FacesServlet)
  2. which processes `foo.xhtml`
  3. and sends the result to the browser

- For a GET request, the bean getters are invoked

Response handling

- Created using a template language, Facelets:
  - XML! ☺
  - Based on XHTML with extra elements in JSF namespace
- JSF runs the template system at every request
- Simple programming language:
  - If-then-else through *panelGroup* element
  - for loops through *repeat* element etc

JSF Facelet tags

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:composite="http://java.sun.com/jsf/composite"
      xmlns:c="http://java.sun.com/jstl/core"
      xmlns:fn="http://java.sun.com/jsp/jstl/functions">
   ...
</html>
```

– see the tag API at http://www.javaserverfaces.org

JSF Tags: repeat

- Used for loops (similar to Java's *for in*) in templates:

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
  <ul>
    <ui:repeat var="o" value="#{mybean.items}" varStatus="status">
       <li>
         #{o}
       </li>
    </ui:repeat>
  </ul>
</ui:composition>
```

JSF Tags: dataTable

- Specialized `repeat` that generates a HTML table:

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
  <ul>
    <h:dataTable var="o" value="#{mybean.items}">
       <h:column>
         #{o}
       </h:column>
    </h:dataTable>
  </ul>
</ui:composition>
```

JSF Tags: panelGroup

- Used for conditional views (if-then-else):

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
  <ul>
    <h:panelGroup rendered="#{mybean.shouldRender}">
      <ui:repeat var="o" value="#{mybean.items}" varStatus="status">
        <li>
          #{o}
        </li>
      </ui:repeat>
    </h:panelGroup>
  </ul>
</ui:composition>
```

JSF Tags: outputText

- Used to control text output.
  - For example, disable output escaping (render text as HTML):

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
  <ul>
    <h:outputText value="#{mybean.text}" escape="false">
  </ul>
</ui:composition>
```

- #{o} in text is shorthand for <h:outputText value="#{mybean.text}">
  - See previous examples

JSF HTML tags

- Many ordinary HTML tags have a JSF HTML counterpart

- `h:form`
- `h:inputText`, `h:inputTextarea`, `h:selectOneListbox`, `h:selectManyCheckbox`
- `h:commandButton`, `h:commandLink`, `h:button`, `h:link`
- `h:head`, `h:body`
- `h:graphicImage`
- `h:panelGrid`
- `h:dataTable`
- …

- All these provide extra functionality over their HTML counterparts. **Use only if you need this extra functionality!**

Facelet composition

- Reuse template parts on multiple pages
  - Reusable components must be well-formed XML snippets

wrapper.xhtml:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Hello!</title>
  </h:head>
  <h:body>
    <ui:insert name="stuff" />
  </h:body>
</html>
```

```
<?xml version='1.0' encoding='UTF-8' ?
<ui:composition xmlns="http://www.w3.c
      xmlns:ui="http://java.sun.com/js
      template="wrapper.xhtml">
  <ui:define name="stuff">
    dWebTek is fun!
  </ui:define>
</ui:composition>
```

The expression language

- Simple language for binding Java properties to Facelets
- Typically used for bean getters and setters
- Also used for invoking bean methods in `action` attributes
- The syntax **`#{exp}`** holds the expression **`exp`**

The Expression Language

- We want to **avoid Java code** in JSF templates
- The syntax **${exp}** may be used in
  - template text
  - attribute values in markup
- The expression may access
  - variables in the various scopes
  - implicit objects, such as *param*: **`#{param.x}`**
  - getters in Java beans: **`#{mybean.name}`**
  - bean methods for *action* attributes
- The usual operators (like +, - etc.) are available

EL – predefined objects

Available in addition to beans:

- `header`
- `param`
- `requestScope`
- `viewScope`
- `sessionScope`
- `applicationScope`
- `resource`
- …

Overview

- Motivation
- Hello World
- Requests and responses
- **Data storage**
- Navigation
- Guessing Game in JSF
- Advanced features
- Deployment

Data storage and Java Beans

- All data is stored in Java beans:
  - Request parameter values
  - Session data
  - Shared application state
- Annotations on beans define the scope

- Java beans also hold 'bean methods':
  - Not getters and setters
  - Invoked from *action* attributes in forms

Bean annotations

- `@ManagedBean(name="someBean")`
- `@ManagedProperty("#{otherBean}")`

- `@RequestScoped` (the default)
- `@ViewScoped` (special to JSF, survives the re-generation of the page with POST response)
- `@SessionScoped`
- `@ApplicationScoped`
- …

- Can also work with Java EE standard bean annotations
- Make session scoped beans `Serializable` to permit clustering

Clicker question

- How would you characterize the way we access the HTTP request data in JSF?
  - The access in implicit in that we access only injected values 
  - The access in explicit through the JSFRequest object
  - We never access HTTP Request data in JSF

Overview

- Motivation
- Hello World
- Requests and responses
- Data storage
- **Navigation**
- Guessing Game in JSF
- Advanced features
- Deployment

Navigation

- JSF handles navigation between pages
  - In Servlets, we remembered URLs and navigated manually
- Two types:
  - Static navigation: Always to the same page
  - Dynamic navigation: Bean decides navigation target
- Parameter values are inserted from forms or param elements.

Navigation in forms

- Static:

```
<h:commandButton action="response" value="Submit"/>
```

- Dynamic:

```
<h:commandButton action="#{foo.bar}" value="Submit"/>
```

```
@ManagedBean
...
public class Foo {
   public String bar() {
     if (...)
       return "success";
     else
       return "failure";
  }
  ...
}
```

```
<faces-config ...>
  ...
  <navigation-rule>
    <from-view-id>/welcome.xhtml</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/response.xhtml</to-view-id>
    </navigation-case>
    ...
  </navigation-rule>
</faces-config>
```

(ensures that the model does not depend on the view!)

Navigation in links

- Static:

```
<h:commandLink action="response” value="Click me!"/>
```

- Dynamic:

```
<h:commandLink action=”#{foo.bar}” value="Click me!"/>
```

- With HTTP parameters:

```
<h:commandLink action=”#{foo.bar}” value="Click me!">
  <f:param name="id" value="#{itemBean.id}" />
<h:commandLink>
```

- Also a <h:link> tag that generates simpler HTML

Interactions between pages

each page submits the form data to itself!

When a request (with POST form data) arrives:

- Restore or create page component tree
- Store request values in component objects (via converters)
- Process validations
- Update model values (invoke bean setters)
- Invoke application (the action) determines where to go next

the ***previous*** page

- Render response

the new page

POST vs. GET

- GET requests just render the page
  - reading GET parameters:
    `f:metadata` and `f:viewParam`
  - making GET links and submit buttons:
    `h:link` and `h:button`
- POST is used for most interactions (as in the example)

- … but POST is not always the right choice
  - not bookmarkable
  - not cacheable

The *POST-redirect-GET* trick

- A solution to the "one step behind" problem
- Also makes POST responses bookmarkable and cacheable

```
<h:commandButton action="response?faces-redirect=true"
                 value="Submit"/>
```

or

```
<navigation-case>
  ...
  <redirect/>
</navigation-case>
```

- Sends a HTTP redirect to the browser, which then makes a GET request to the new page

Clicker question

- How do we store client-specific data in JSF?
  - In a @SessionScoped bean 
  - In the JSFSession object
  - In the template code

Overview

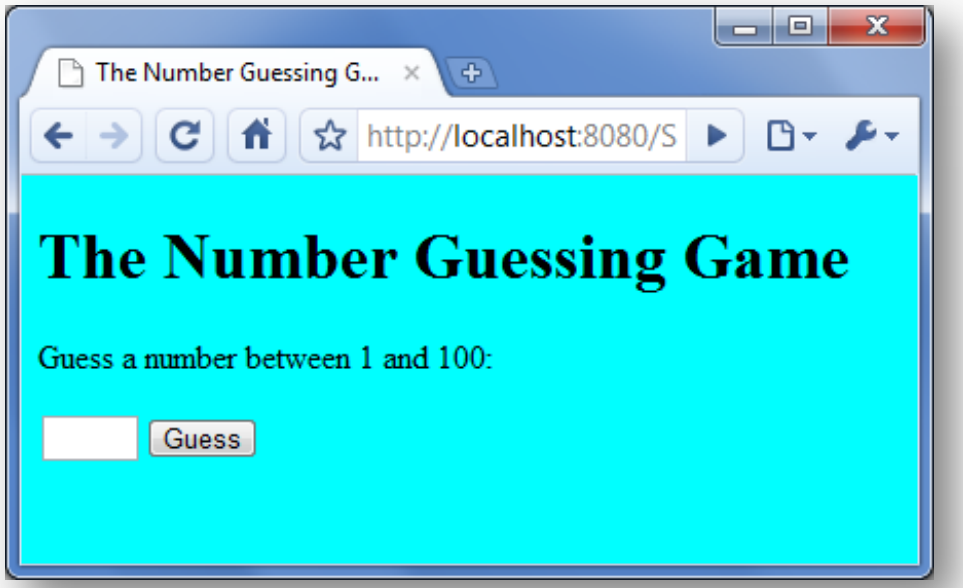
- Motivation
- Hello World
- Requests and responses
- Data storage
- Navigation
- **Guessing Game in JSF**
- Advanced features
- Deployment

Example: the guessing game

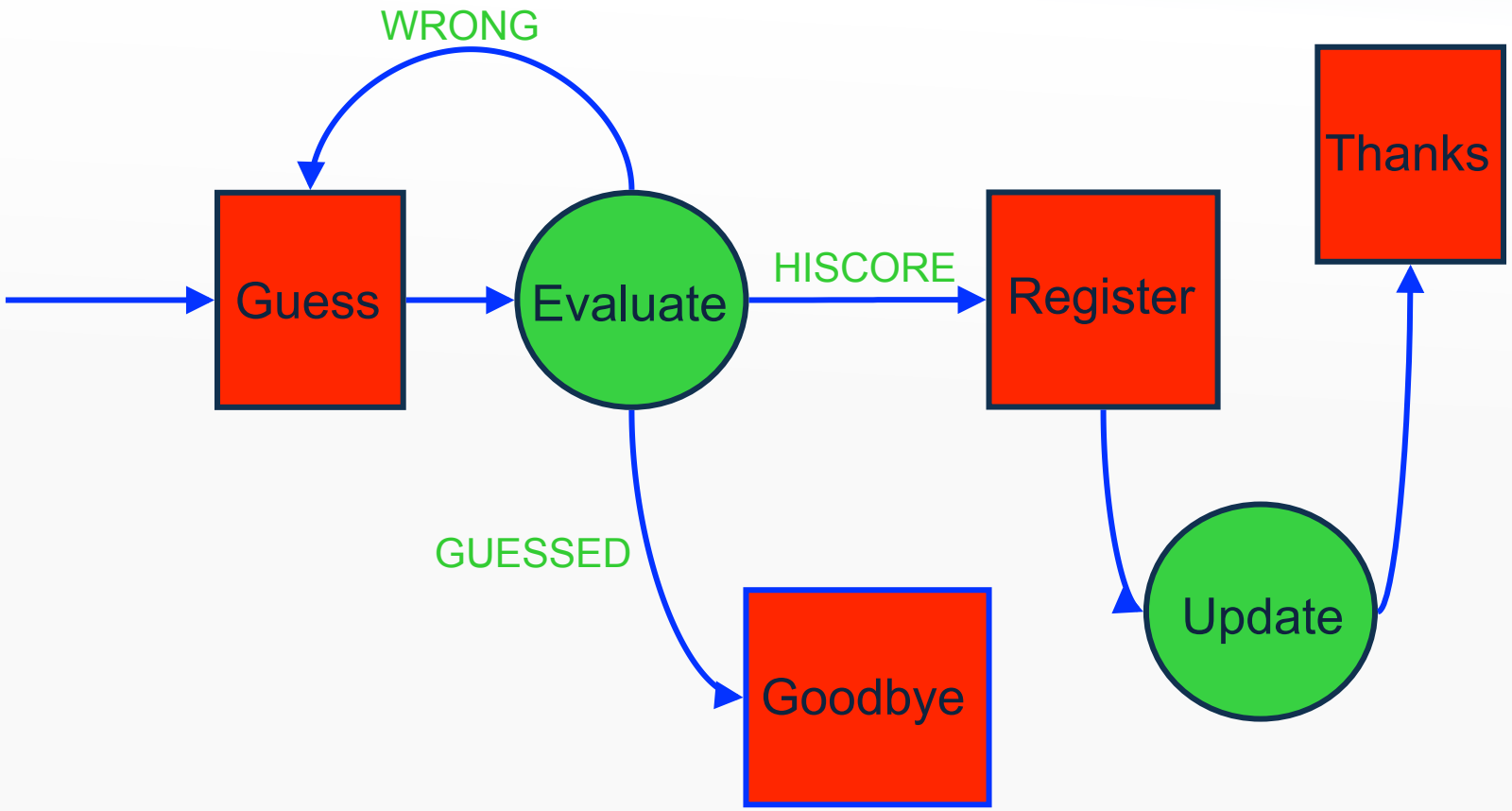
- web/WEB-INF/web.xml
- web/WEB-INF/faces-config.xml

- src/dwebtek/Hiscore.java
- src/dwebtek/GameSession.java

- web/template.xhtml
- web/hiscore.xhtml
- web/guess.xhtml
- web/goodbye.xhtml
- web/register.xhtml
- web/thanks.xhtml
- web/style.css



Number Guessing Game



faces-config.xml (1/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
    version="2.0">

    <navigation-rule>
      <from-view-id>/guess.xhtml</from-view-id>
      <navigation-case>
        <from-outcome>WRONG</from-outcome>
        <to-view-id>/guess.xhtml</to-view-id>
      </navigation-case>
      <navigation-case>
        <from-outcome>HISCORE</from-outcome>
        <to-view-id>/register.xhtml</to-view-id>
        <redirect/>
      </navigation-case>
      <navigation-case>
        <from-outcome>GUESSED</from-outcome>
        <to-view-id>/goodbye.xhtml</to-view-id>
        <redirect/>
      </navigation-case>
    </navigation-rule>
```

faces-config.xml (2/2)

```
    <navigation-rule>
      <from-view-id>/register.xhtml</from-view-id>
      <navigation-case>
        <from-outcome>DONE</from-outcome>
        <to-view-id>/thanks.xhtml</to-view-id>
        <redirect/>
      </navigation-case>
    </navigation-rule>
    <navigation-rule>
      <from-view-id>/thanks.xhtml</from-view-id>
      <navigation-case>
        <from-outcome>AGAIN</from-outcome>
        <to-view-id>/guess.xhtml</to-view-id>
        <redirect/>
      </navigation-case>
    </navigation-rule>
    <navigation-rule>
      <from-view-id>/goodbye.xhtml</from-view-id>
      <navigation-case>
        <from-outcome>AGAIN</from-outcome>
        <to-view-id>/guess.xhtml</to-view-id>
        <redirect/>
      </navigation-case>
    </navigation-rule>

</faces-config>
```

Hiscore.java (1/2)

```
package dwebtek;

import javax.faces.bean.ApplicationScoped;
import javax.faces.bean.ManagedBean;

@ManagedBean
@ApplicationScoped
public class Hiscore {

    private int plays;
    private String holder;
    private int record;

    public synchronized void newPlay() {
        plays++;
    }
```

Hiscore.java (2/2)

```
    public synchronized void setHiscore(String name, int guesses) {
        if (holder == null || guesses < record) {
            holder = name;
            record = guesses;
        }
    }

    public int getPlays() { return plays; }
    public void setPlays(int plays) { this.plays = plays; }

    public String getHolder() { return holder; }
    public void setHolder(String holder) { this.holder = holder; }

    public int getRecord() { return record; }
    public void setRecord(int record) { this.record = record; }
}
```

GameSession.java (1/3)

```
package dwebtek;

import java.io.Serializable;
import java.util.Random;

import javax.annotation.PostConstruct;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.SessionScoped;

@ManagedBean(name="game")
@SessionScoped
public class GameSession implements Serializable {

    private int secret;
    private int guesses;
    private Integer number;
    private String name;
    private String message;

    @ManagedProperty("#{hiscore}")
    transient private Hiscore hiscore;
```

GameSession.java (2/3)

```
@PostConstruct
public void init() {
     secret = new Random().nextInt(100) + 1;
     guesses = 0;
     number = null;
     setMessage("Guess a number between 1 and 100");
     hiscore.newPlay();
}

public String makeGuess() {
     guesses++;
     if (number > secret) {
         setMessage("Your guess is too large, guess again");
         return "WRONG";
     } else if (number < secret) {
         setMessage("Your guess is too small, guess again");
         return "WRONG";
     } else if (hiscore.getHolder() == null ||
                guesses < hiscore.getRecord()) {
         return "HISCORE";
     } else {
         return "GUESSED";
     }
}
```

GameSession.java (3/3)

```
    public String register() {
        hiscore.setHiscore(name, guesses);
        return "DONE";
    }

    public String again() {
        init();
        return "AGAIN";
    }

    public Integer getNumber() { return number; }
    public void setNumber(Integer number) { this.number = number; }
    public int getSecret() { return secret; }
    public void setSecret(int secret) { this.secret = secret; }
    public int getGuesses() { return guesses; }
    public void setGuesses(int guesses) { this.guesses = guesses; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message;}
    public void setHiscore(Hiscore hiscore) { this.hiscore = hiscore; }
}
```

template.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:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <meta http-equiv="cache-control" content="no-store"/>
    <link href="style.css" rel="stylesheet" type="text/css" />
    <title>The Number Guessing Game</title>
  </h:head>
  <h:body>
    <h1>The Number Guessing Game</h1>
    <ui:insert name="content" />
  </h:body>
</html>
```

hiscore.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:c="http://java.sun.com/jsp/jstl/core"
      template="template.xhtml">
  <ui:define name="content">
    <c:choose>
      <c:when test="#{hiscore.record gt 0}">
        In #{hiscore.plays} plays of this game,
        the record holder is #{hiscore.holder} with
        <b>#{hiscore.record}</b> guesses.
      </c:when>
      <c:otherwise>
        No winners yet.
      </c:otherwise>
    </c:choose>
  </ui:define>
</ui:composition>
```

guess.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:c="http://java.sun.com/jsp/jstl/core"
      template="template.xhtml">
  <ui:define name="content">
    <p>
      #{game.message}:
    </p>
    <h:form>
      <h:inputText size="2" maxlength="2" value="#{game.number}"/>
      <h:commandButton value="Guess" action="#{game.makeGuess}"/>
    </h:form>
  </ui:define>
</ui:composition>
```

goodbye.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:c="http://java.sun.com/jsp/jstl/core"
      template="template.xhtml">
  <ui:define name="content">
    <p>
      You used #{game.guesses} guesses.
    </p>
    <p>
      #{hiscore.holder} has the highscore with
      #{hiscore.record} guesses.
    </p>
    <h:form>
      <h:commandLink action="#{game.again}">Play again</h:commandLink>
    </h:form>
  </ui:define>
</ui:composition>
```

register.xhtml

```
<c:choose xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:c="http://java.sun.com/jsp/jstl/core">
  <c:when test="${game.number != game.secret}">
    <ui:include src="thanks.xhtml"/>
  </c:when>
  <c:otherwise>
    <ui:decorate template="template.xhtml">
      <ui:define name="content">
        <p>
          You have the new highscore of
          #{game.guesses} guesses!
        </p>
        <p>
          Please enter your name:
        </p>
        <h:form>
          <h:inputText value="#{game.name}"/>
          <h:commandButton value="Register" action="#{game.register}"/>
        </h:form>
     </ui:define>
    </ui:decorate>
  </c:otherwise>
</c:choose>
```

thanks.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:c="http://java.sun.com/jsp/jstl/core"
      template="template.xhtml">
  <ui:define name="content">
    <p>
      Thanks for playing.
    </p>
    <h:form>
      <h:commandLink action="#{game.again}">Play again</h:commandLink>
    </h:form>
  </ui:define>
</ui:composition>
```

style.css

```
body {
  background-color: aqua;
}
```

web.xml (1/2)

```
<?xml version="1.0"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                             http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">
  <display-name>The Number Guessing Game</display-name>
  <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>*.jsf</url-pattern>
  </servlet-mapping>
```

web.xml (2/2)

```
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>XHTML</web-resource-name>
      <url-pattern>*.xhtml</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <description>Prevents access to source files</description>
    </auth-constraint>
  </security-constraint>
  <welcome-file-list>
    <welcome-file>guess.jsf</welcome-file>
  </welcome-file-list>

</web-app>
```

Clicker question

- Where do we use Java code in JSF?
  - For generating XHTML
  - For handling client input 
  - For configuring the application flow

Overview

- Motivation
- Hello World
- Requests and responses
- Data storage
- Navigation
- Guessing Game in JSF
- **Advanced features**
- Deployment

Input validation

- Validator can ensure that input respects a format:

```
@FacesValidator("dk.au.cs.dwebtek.FooValidator")
public class FooValidator implements Validator {
  public void validate(FacesContext context, UIComponent component,
                       Object value) throws ValidatorException {
        if (!value.equals("foo")) {
            throw new ValidatorException(new FacesMessage("Value must
be foo!"));
        }
    }
}
```

```
<h:inputText name="fieldName" ...>
    <f:validator validatorId="dk.au.cs.dwebtek.FooValidator" />
</h:inputText>
<h:message for="fieldName" />
```

Custom tags 1/2

- We can create new tags!

```
@FacesComponent("dk.au.cs.dwebtek.hello")
public class HelloComponent extends UIComponentBase {
    public String getFamily() {return "dk.au.cs.dwebtek.hello";}

    public void encodeBegin(FacesContext arg0) throws IOException {
        super.encodeBegin(arg0);
    }

    public void encodeEnd(FacesContext context) throws IOException {
        ResponseWriter writer = context.getResponseWriter();
        writer.startElement("div", this);
        writer.write("Hello World");
        writer.endElement("div");
    }
}
```

Custom tags 2/2

WEB-INF/custom-taglib.xml:

```
<facelet-taglib version="2.0"
                xmlns="http://java.sun.com/xml/ns/javaee">
    <namespace>http://cs.au.dk/dwebtek/</namespace>
    <tag>
        <tag-name>hello</tag-name>
        <component>
            <component-type>dk.au.cs.dwebtek.hello</component-type>
        </component>
    </tag>
</facelet-taglib>
```

In web.xml:

```
<context-param>
        <param-name>javax.faces.FACELETS_LIBRARIES</param-name>
        <param-value>/WEB-INF/custom-taglib.xml</param-value>
</context-param>
```

Using our new tag

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:w="http://cs.au.dk/dwebtek/">
  <head><title>Response</title></head>
  <body>
     <w:hello />
  </body>
</html>
```

- Custom tag attributes:
  - Receive in Java through `getAttributes`

Converters 1/2

- Add new supported classes to input/output in JSF forms:

```
@FacesConverter("dk.au.cs.dwebtek.Converter")
public class SomeConverter implements Converter {
     public Object getAsObject(FacesContext facesContext,
                               UIComponent uiComponent, String s) {
        //Convert from String to Object
     }


     public String getAsString(FacesContext facesContext,
                               UIComponent uiComponent, Object o) {
        //Convert from Object to String
     }
}
```

Converters 2/2

- Using the converter in a form:

```
...
<h:form>
  <h:inputText ...>
    <f:converter converterId="dk.au.cs.dwebtek.Converter"/>
  </h:inputText>
</h:form>
...
```

Clicker question

- If you were to write XML to the JSF output, what would you use?
  - A custom tag 
  - Java code embedded in the template code
  - A converter

Features not covered here

- Event handlers
- Properties files and internationalization
- Lots of libraries
- Ajax
- …

- A decent tutorial: http://docs.oracle.com/javaee/7/tutorial/doc/jsf-page.htm#BNAQZ

Overview

- Motivation
- Hello World
- Requests and responses
- Data storage
- Navigation
- Guessing Game in JSF
- Advanced features
- **Deployment**

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">
  <display-name>MyHelloWorld</display-name>
  <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>*.jsf</url-pattern> <!-- common alternative: faces/* -->
  </servlet-mapping>
  <context-param> <!-- optional, gives more debug info -->
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
</web-app>
```

Getting started

Jar files: (place in e.g. `WEB-INF/lib`)

- `javax.faces-2.2.1.jar`
  in Mojarra (the JSF reference implementation)
  http://javaserverfaces.java.net/download.html (current release, binary)

If using JSTL function library: http://jstl.java.net/download.html

- `javax.servlet.jsp.jstl-api-1.2.1.jar`
- `javax.servlet.jsp.jstl-1.2.1.jar`

To use latest Expression Language:

- `el-api-2.2.jar` http://download.java.net/maven/2/javax/el/el-api/
- `el-impl-2.2.jar` http://download.java.net/maven/2/org/glassfish/web/el-impl/

`web.xml`:

```
<context-param>
  <param-name>com.sun.faces.expressionFactory</param-name>
  <param-value>com.sun.el.ExpressionFactoryImpl</param-value>
</context-param>
```

Overview

- Motivation
- Hello World
- Requests and responses
- Data storage
- Navigation
- Guessing Game in JSF
- Advanced features
- Deployment

Some observations...

- Clean separation of M, V, and C
- The example code is mostly "declarative" in nature
- Form state automatically stored in beans

- **The page that produces a form also takes care of receiving the form state!**
  - We get a "one step behind" mismatch between the URL in the browser and the page being shown (try it and see!)

Online resources

- Oracle's JSF page: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
- Tutorial: http://www.oracle.com/technetwork/java/javaee/documentation/index-137726.html