Java Servlets und Java Server Pages

JavaServlets und JavaServer Pages

Servlet und Servel-Container

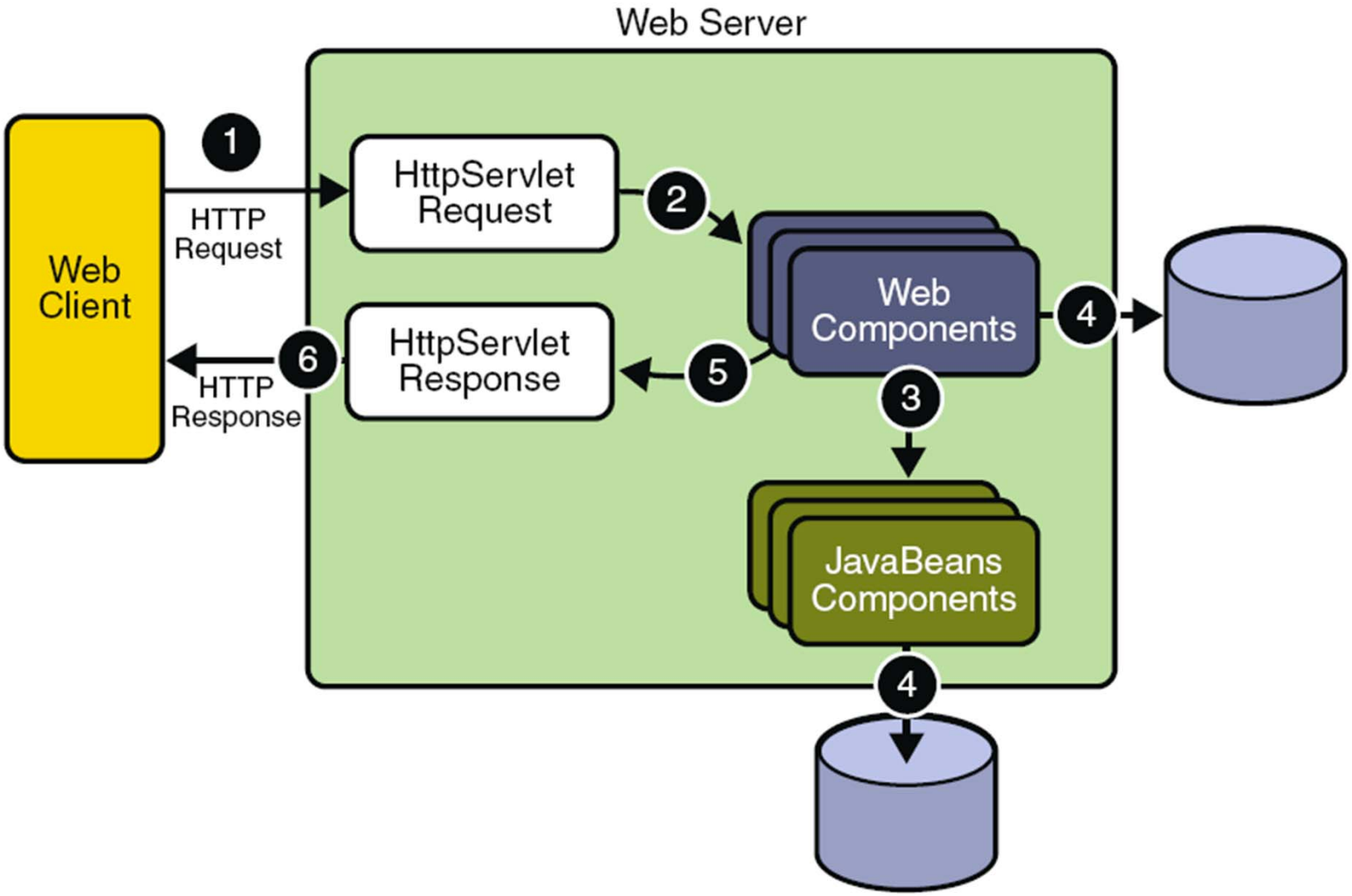
Servlet

- Eine auf Java-Technologie basierte Web-Komponente, die von einem Servlet-Container verwaltet wird und dynamisch Inhalt generiert.

Servlet-Container

- Teil eines Web-Servers, der die Netzwerkdienste zum Empfangen von Anfragen und Senden von Antworten bereitstellt und die Servlets über ihren gesamten Lebenszyklus enthält und verwaltet.

Design


Web Server
Web Client
1
HTTP Request
HttpServlet Request
2
Web Components
4
3
JavaBeans Components
4
5
HttpServlet Response
6
HTTP Response

JavaServlets und JavaServer Pages

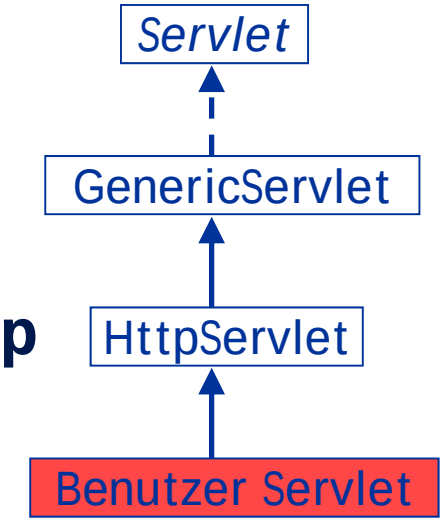
API

Generische Klassen: javax.servlet

- Servlet ← GenericServlet
- ServletRequest
- ServletResponse
- ServletContext
- …

Spezialisierungen für HTTP: javax.servlet.http

- HttpServlet
- HttpServletRequest
- HttpServletResponse
- HttpSession
- …



Servlet Interface

`void init(ServletConfig config)`
`throws ServletException`

- Initialisierung des Servlets mit Konfigurationsparameter

`ServletConfig getServletConfig()`

- Zugriff auf die Konfigurationsparameter

`void service(ServletRequest req, ServletResponse res)`
`throws ServletException, IOException`

- Führt einen einzelnen Request von einem Client aus und erzeugt die Response.
- Für jeden Request wird die Methode in eigenem Thread ausgeführt
- Parameter **`req`** hält alle Informationen über Request;
- in Parameter **`res`** wird die Antwort geschrieben

`String getServletInfo()`

- Information über Servlet in Form eines Strings

`void destroy()`

- soll Aufräumarbeiten durchführen und zerstört das Servlet

HttpServlet

protected Methoden:

- `void doDelete(HttpServletRequest, HttpServletResponse)`
- `void doGet(HttpServletRequest, HttpServletResponse)`
- `void doHead(HttpServletRequest, HttpServletResponse)`
- `void doOptions(HttpServletRequest, HttpServletResponse)`
- `void doPost(HttpServletRequest, HttpServletResponse)`
- `void doPut(HttpServletRequest, HttpServletResponse)`
- `void doTrace(HttpServletRequest, HttpServletResponse)`
- `void service(HttpServletRequest, HttpServletResponse)`
  - Leitet HTTP-Anfragen auf die entsprechenden Methoden um.

public Methoden:

- `void service(ServletRequest, ServletResponse)`
  - Ruft die `protected service`-Methode auf

HttpServletResponse

`ServletOutputStream getOutputStream()`

- Binäre Daten

`PrintWriter getWriter()`

- Text

exklusiv für Response

`void reset()`

- Löschen des Puffers, der Header und des Statuscodes

`void flushBuffer()`

- Schreiben des Puffers (den Header und Statuscode) an den Client

`String getContentType()`

- Liefert den aktuell gesetzten MIME Typen (z.B.: text/html, image/jpeg, ...)

`void setContentType(String type)`

- Setzt den MIME Typen (z.B.: text/html, image/jpeg, ...)

`...`

Beispiel

```
package TestPack;

import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {
  public void doGet (HttpServletRequest request,
                     HttpServletResponse respose)
  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(
      "<html><head><title>Hello World</title></head><body>"
      + "<h2>Hello World!</h2>"
      + "</body></html>"
    );
    out.close();
  }
}
```

HTTP Protokoll

HTTP methods

GET

HEAD

POST

PUT

DELETE

TRACE

CONNECT

OPTIONS

Parameter als Name/Wert-Paare in URL

```
GET /MyServlet.do?name=Max%20Muster&company=MuCom HTTP/1.1
Host: 140.78.145.103:2020
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US;
  rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1
Accept:
  text/xml,application/xml,application/xhtml+xml,text/html;
  q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

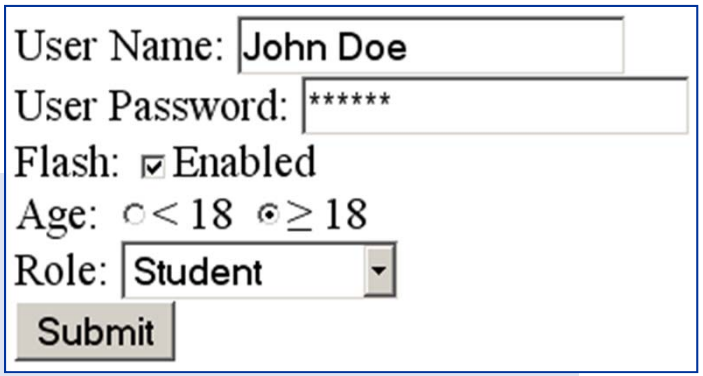
```
POST /MyServlet HTTP/1.1
Host: 140.78.145.103:2020
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US;
  rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1
Accept:
  text/xml,application/xml,application/xhtml+xml,text/html;
  q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 29

name=Max+Muster&company=MuCom
```

Parameter in eigenem Datenbereich

HTML Formulare

Ermöglichen Benutzereingaben

Benutzen POST oder GET




```
<html><head><title>HTML Forms</title></head>
<body>
<form action="result.html" method="get">
  <input type="hidden" name="id" value="123"/>
  User Name: <input type="text" name="user"/><br>
  User Password: <input type="password" name="pass"/><br>
  Flash:
  <input type="checkbox" name="flash" checked>Enabled</input><br>
  Age:
  <input type="radio" name="age" value="child">&lt; 18</input>
  <input type="radio" name="age" value="adult" checked>&ge; 18</input><br>
  Role:
  <select name="role">
    <option value="A">Administrator</option>
    <option value="S" selected>Student</option>
  </select><br>
  <input type="submit" value="Submit"/>
</form>
</body></html>
```

Aufruf:

```
result.html?id=123&user=John+Doe&pass=mypass&flash=on&age=adult&role=S
```

HTML Formulare Fortsetzung

`id=123&user=John+Doe&pass=mypass&flash=on&age=adult&role=S`

Textfelder:

- **`text`**: Text-Eingabefeld
- **`hidden`**: Unsichtbares Textfeld, Information für den Server (z.B.: AppointmentId)
- **`password`**: Text-Einfabefeld bei dem die Eingabe versteckt wird

Sonderzeichen werden codiert übertragen (%HexHex; z.B.: ä=%E4),

Leerzeichen als "+" (z.B.: John+Dow).

Auswahlfelder:

- **`checkbox`**: Wert wird nur übertragen wenn er gesetzt ist (als name=on, z.B.: flash=on), mit **`checked`** kann eine Checkbox vorselektiert werden.
- **`radio`**: Wert wird nur übertragen wenn er gesetzt ist (value wenn angegeben sonst "on"), es kann nur ein Wert aus einer Gruppe (gleicher Name) ausgesucht werden, mit **`checked`** kann eine Vorselektion erfolgen.
- **`select`**: Dropdown-Element, als Wert wird der ausgewählte (value der Option wenn angegeben sonst der Text) übertragen wobei mit **`selected`** eine Vorauswahl getroffen werden kann (sonst erstes Element).

Absenden:

- **`submit`**: Erzeugt einen Button, Text kann mit **`value`** gesetzt werden, wird auch ein **`name`** gegeben so wird das Name/Wert-Paar mitübertragen (sinnvoll wenn mehrere Buttons in einem Formular).

HttpServletRequest

`String getParameter(String name)`

- Wert (URL-decoded) des ersten Parameters mit dem gegebenen Namen
- **`null`**, falls nicht vorhanden

`String[] getParameterValues(String name)`

- Array mit Werten (URL-decoded) für jedes Vorkommen des angegebenen Parameternamens
- **`null`**, falls nicht vorhanden

`Enumeration request.getParameterNames()`

- eine Aufzählung (*Enumeration*) aller Parameternamen
- falls keine Parameter übergeben wurden, ist die Enumeration leer

`String getRemoteAddress()`

- IP-Adresse des Clients

`boolean isSecure()`

- **`true`**, wenn der Aufruf über HTTPS erfolgt ist

`...`

Beispiel Servlet: Addieren von 2 Parameterwerten

```
package calc;
@WebServlet("/AdderServlet")
public class AdderServlet extends HttpServlet {
  ...
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    String aStr = request.getParameter("a");
    int a = 0;
    if (aStr != null) {
      try { a = Integer.parseInt(aStr);  } catch (N
    }
    String bStr = request.getParameter("b");
    int b = 0;
    if (bStr != null) {
    try { b = Integer.parseInt(bStr } catch (Number
    int ab = a + b;
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>Adder</title></head>\n<body>");
    out.println("<h1>Add a and b</h1>");
    out.println("<form action=\"AdderServlet\" method=\"get\"/><br>");
    out.println("a =  <input type=\"text\" name=\"a\" value=\"" + a + "\"/><br>");
    out.println("b =  <input type=\"text\" name=\"b\" value=\"" + b + "\"/><br>");
    out.println("a + b =  " + ab + "<br>");
    out.println("<input type=\"submit\" value=\"Add\"/>");
    out.println("</form>");
    out.println("</body>\n</html>");
    out.close();
  }
  ...
}
```

form

Parameter a und b

JavaServlets und JavaServer Pages

- Einführung
- Servlets
- **Lebenszyklus und Sessions**
- Installation
- JavaServer Pages
- Trennung Logik und Darstellung
- Diverses
- Zusammenfassung

Lebenszyklus

Engine lädt Servlet-Klasse und erzeugt ein Exemplar

- alle Variablen werden genau einmal initialisiert
- alle Variablen bleiben aktiv, solange das Servlet-Exemplar existiert

Mehrere Web-Browser (=Clients) fordern das Servlet an

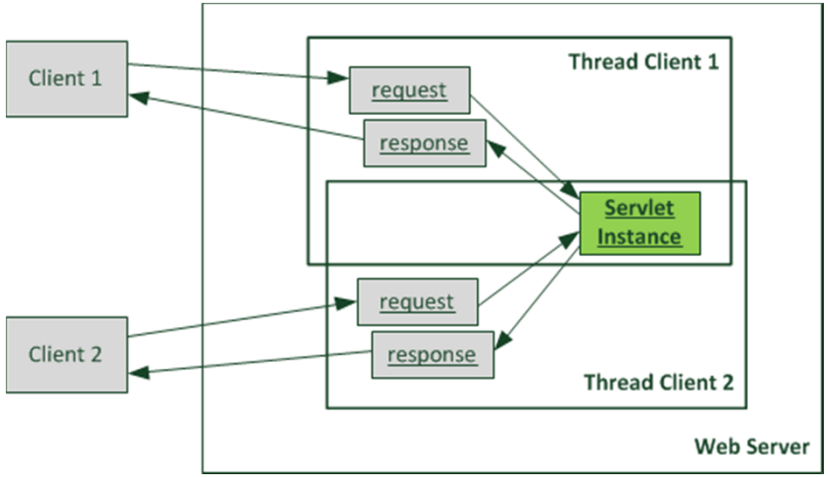
- Request wird vom Web-Server an Servlet-Engine und von dieser weiter zum entsprechenden Servlet gesandt

Engine erzeugt pro Request einen Thread

- jeder Thread behandelt Request+Response des jeweiligen Clients
- jeder Thread hat Zugriff auf die Instanzvariablen des Servlets

Servlet-Engine bestimmt, wann Servlets entladen werden

- Servlet gibt Ressourcen frei und speichert persistenten Zustand



Lebenszyklus

`init(ServletConfig)`

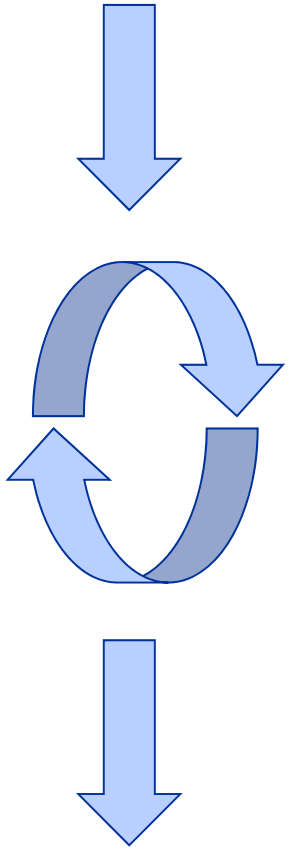
- Variablen initialisieren, Ressourcen anfordern

`service(HttpRequest, HttpResponse)`

- `doGet(HttpRequest, HttpResponse)`
- `doPost(HttpRequest, HttpResponse)`
- `doPut(HttpRequest, HttpResponse)`
- `...`

`destroy()`

- Ressourcen freigeben
- Eventuell Zustand speichern



Sitzungen (Sessions)

Probleme

- HTTP ist ein nicht sitzungsorientiertes Protokoll
  - Wenn der eBusiness-Kunde einen Artikel in den Einkaufskorb legt, woher weiß der Server, was schon im Einkaufskorb ist?
  - Wenn der Kunde seine Online-Einkaufstour beendet, woher weiß der Server, welcher Einkaufskorb zu welchem Kunden gehört?

Lösungen

- Cookies
- URL mit Parameter
- Versteckte HTML-Formular-Felder

Unterstützung

- Servlets bieten ein „Higher Level API“
- *HttpSession*

Sitzungen (Sessions) API

HttpSession

- ***HttpSession*** **<HttpServletRequest>.getSession()**
  - Liefert das aktuelle Session-Objekt, oder erzeugt ein neues, wenn noch keines existiert.
- ***HttpSession*** **<HttpServletRequest>.getSession(boolean c)**
  - c = true: neues Session-Objekt anlegen, wenn noch keines existiert.
  - c = false: null liefern, wenn noch kein Session-Objekt existiert.
- **Object getAttribute(String name)**
  - Liefert das Objekt mit dem angegebenen Namen oder null, falls es nicht existiert.
- **void setAttribute(String name, Object o)**
  - Speichert das gegebene Objekt unter dem gegebenen Namen.
  - Existiert ein Name bereits, wird das Objekt überschrieben
  - Existiert ein Name bereits und o==null, so wird das Objekt entfernt

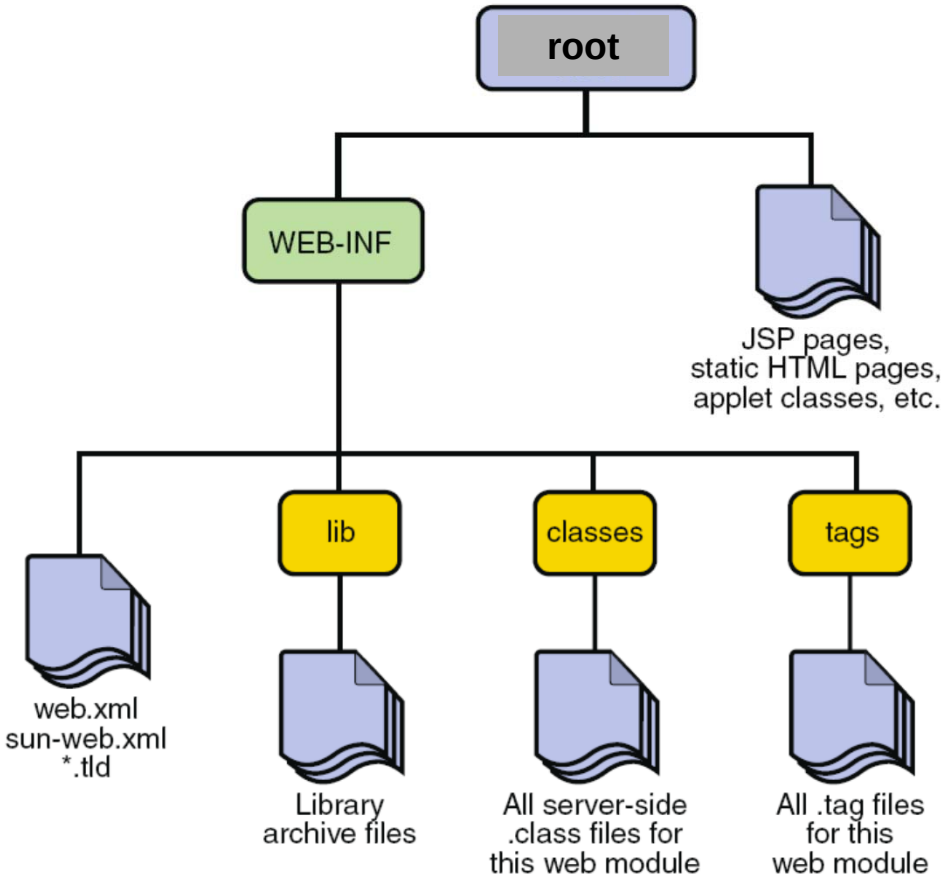
Beispiel Sessions: Summieren von Werten

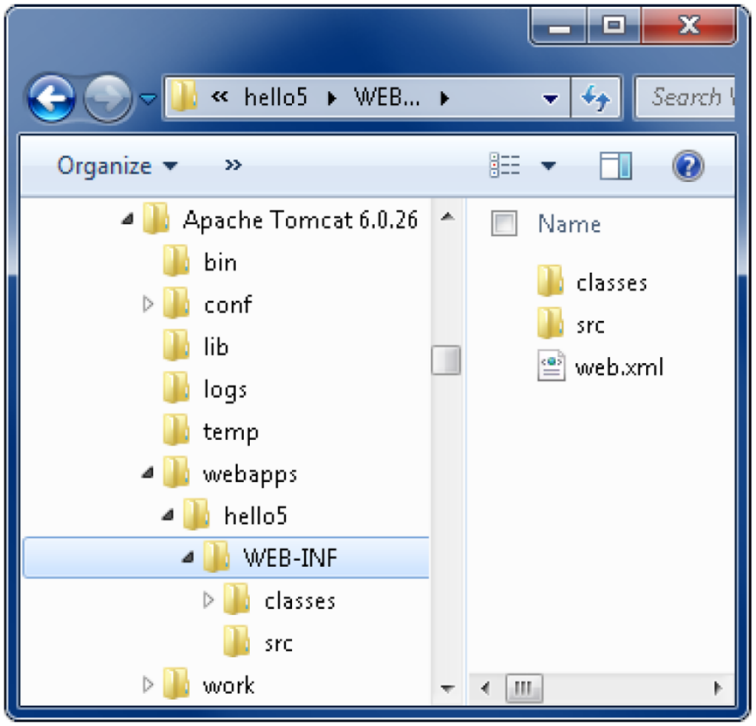
```
package calc;

@WebServlet("/SumServlet")
public class SumServlet extends HttpServlet {
  ...
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    String valueStr = request.getParameter("value");
    int value = 0;
    if (valueStr != null) {
      try { value = Integer.parseInt(valueStr); } catch (NumberFormatException e) { }
    }

    HttpSession session = request.getSession();
    Integer sum = (Integer) session.getAttribute("sum");
    if (sum == null) sum = 0;
    sum = sum + value;
    session.setAttribute("sum", sum);

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><head><title>Summer<
      out.println("<h1>Sum up values</h1>");
      out.println("<form action=\"SumServlet\
      out.println("next value =  <input type=\
      out.println("sum =  " + sum + "<br>");
      out.println("<input type=\"submit\" val
      out.println("</form>");
      out.println("</body>\n</html>");
      out.close();
  }
  ...
```

Session atrribute sum

JavaServlets und JavaServer Pages

Verzeichnisstruktur am WebServer





Installieren eines Servlets (web.xml 1/3)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <!– Allgemeine Beschreibung -->
    <display-name>Name der Anwendung</display-name>
    <description>Beschreibung der Anwendung</description>
    <!– Kontext Parameter die man aus dem Servlet (oder der JSP Seite) über
          String name="…", param;
          param = getServletContext().getInitParameter(name);
          abfragen kann.
    -->
    <context-param>
      <param-name>Name des Parameters</param-name>
      <param-value>Wert des Parameters</param-value>
      <description>The Beschreibung des Parameters</description>
    </context-param>
 …
```

Installieren eines Servlets (web.xml 2/3)

```
…
  <!– Beschreibung der Servlets dieser Web Anwendung.
       Jedem Servlet können Parameter mitgegeben werden die über
       String name="…", value;
       value = getServletConfig().getInitParameter(name);
       abgefragt werden können.
  -->
  <servlet>
    <servlet-name>Interner Name des Servlets</servlet-name>
    <description>Beschreibung des Servlets</description>
    <servlet-class>Voll qualifizierter Klassenname</servlet-class>
    <init-param>
      <param-name>Name des Parameters</param-name>
      <param-value>Wert des Parameters</param-value>
    </init-param>
    <!–Servlet automatisch mit Tomcat starten, die Zahl gibt die Reihenfolge
         an in der die Servlets gestartet werden. Wenn keine spezielle
         Reihenfolge benötigt wird genügt auch "<load-on-startup/>"
    -->
    <load-on-startup>Zahl</load-on-startup>
  </servlet>
      …
```

Installieren eines Servlets (web.xml 3/3)

```
    <!– Festlegen unter welchem Namen das Servlet erreichbar sein soll. -->
    <servlet-mapping>
      <servlet-name>Interner Name des Servlets</servlet-name>
      <url-pattern>Offizieller Name des Servlets</url-pattern>
    </servlet-mapping>

    <!– Festlegen des Standard-Timeouts einer Sitzung in Minuten.
          Kann vom Servlet über
          HttpSession ses=…;
          int ses.getMaxInactiveInterval();  abgefragt und über
          void setMaxInactiveInterval(int); verändert werden
          verändert werden.
    -->
    <session-config>
      <session-timeout>Minuten</session-timeout>
    </session-config>

</web-app>
```

Installieren eines Servlets (web.xml Beispiel)

```
<web-app>
  <display-name>SWE2</display-name>
  <description>Lecture SWE2, Institute SSW</description>
  <servlet>
      <servlet-name>SumServlet</servlet-name>
      <description>Simple sum sample.</description>
      <servlet-class>calc.SumSerlet</servlet-class>
      <load-on-startup/>
    </servlet>
    <servlet-mapping>
      <servlet-name>SumServlet</servlet-name>
      <url-pattern>/Sum</url-pattern>
    </servlet-mapping>
    <session-config>
      <session-timeout>30</session-timeout>    <!-- 30 minutes -->
    </session-config>
</web-app>
```

JavaServlets und JavaServer Pages

Java Server Pages (JSP)

Template-Sprache mit

- HTML als Template
- eingebetteter Java-Code zur dynamischen Erzeugung des Inhalts

JSP-Datei wird vom WebServer in Servlet übersetzt

JSP vs. Servlet

Servlets

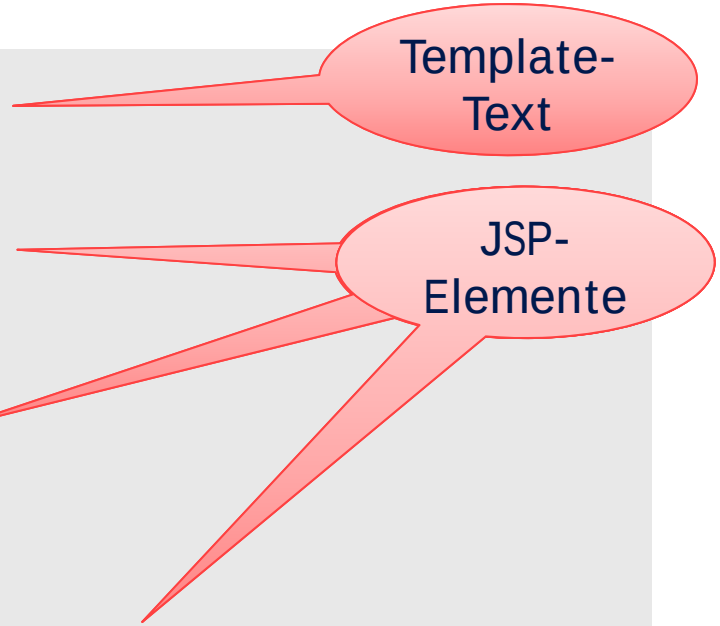
- Java-Programm, das HTML-Text auf OutputStream schreibt

JSP

- HTML-Code kann wie gewohnt verwendet werden
- Java-Scriptlets eingebettet
- damit strikte Trennung zwischen Erscheinungsbild (HTML) und Logik (Java)
- damit Veränderung des Aussehens ohne Veränderung des dynamischen Inhalts

JSP Beispiel

```
<html>
<head><title>JSP 1</title></head>
<body>

<%@ page import="java.util.*, java.text.*" %>

<% int max = 100;  // pure Java code! %>

<P> JSP Test - Zähler bis <%= max %> </P>

<% for (int i = 0; i < max; i++) { %>
       <%= i %>
<% }
   String dateString = DateFormat.getDateInstance(DateFormat.SHORT,
                  Locale.GERMAN).format(new Date());
%>

<P> Heutiges Datum: <%= dateString %> </P>
</body>
</html>
```



JSP Syntax

Direktiven

- <%@ … %>: Eigenschaften & Includes
  - Erlaubt die Einstellung seitenbezogener Optionen.
  - Kann mehrmals vorkommen.
  - Inkludiert eine externe Datei.

```
<%@ page
        language="java"
        import="java.util.*, java.text.*"
        contextType="text/html"
        ...
%>
```

```
<%@ include file="eineDatei.inc" %>
```

```
<%@ taglib
        uri="/tagliburi"
        prefix="kurzname"
%>
```

JSP Syntax

Skriptelemente:

- <% ... %> - Skriptlet
  - Beliebiger ausführbarer Java-Code (Anweisungen, Variablendeklaration).
- <%! ... %> - Deklarationen
  - Variablen- und Methodendeklarationen
- <%= ... %> - Ausdrücke
  - Ausdruck wird in einen String konvertiert und in den Seitentext eingefügt.
  - Beispiele: <%= i%>, <%= new Date()%>, <%= methode()%>
  - Achtung: kein abschließendes Semikolon!

Kommentare:

- <%-- ... --%> - Kommentar
  - Nur in der JSP Datei enthalten, nicht aber im Ergebnis.
- In Skriptelementen normale Java Kommentare (/* */ und //)

Beispiel JSP: Summieren von Werten

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" import="java.util.List" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Sum up values JSP</title>
</head>
<body>
<h1>Sum up values</h1>
<%!
  private static int parseInt(String str) {
    if (str != null) {
      try {
        return  Integer.parseInt(str);
        } catch (NumberFormatException e) {
          return 0;
        }
    } else {
      return 0;
    }
  }
%>
<%
  String valueStr = request.getParameter("value");
  int value = parseInt(valueStr);
  Integer sum = (Integer) session.getAttribute("sum");
  if (sum == null) sum = 0;
  sum = sum + value;
  session.setAttribute("sum", sum);
%>
...
```

Direktive

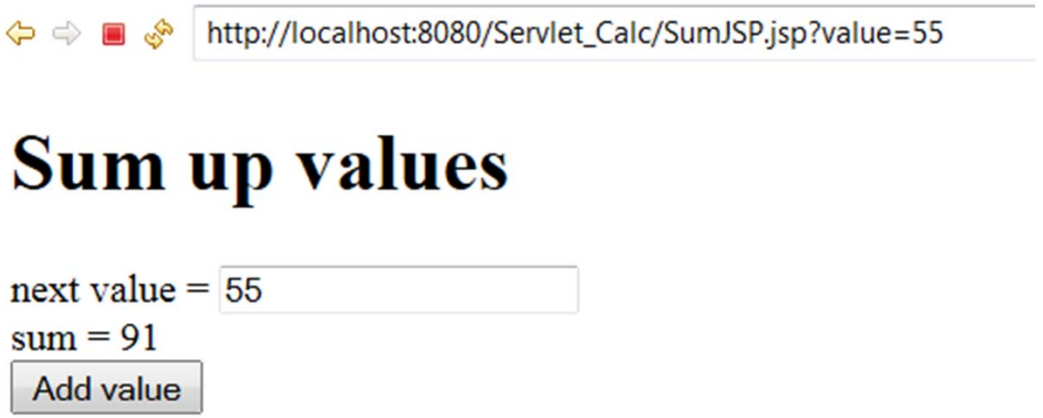
Deklaration

Scriptlet

Beispiel JSP: Summieren von Werten

```
...
<form>
next value =  <input type="text" name="value" value="<%= value %>"/>  <br>
sum = <%= sum %>  <br>
<input type="submit" value="Add value"/>
</form>
</body>
</html>
```

Ausdrücke



JSP Implizite Objekte

Objekt	Scope
ServletRequest: request	Request
ServletResponse: response	Page
▪ Wird normalerweise nicht benutzt.	
PageContext: pageContext	Page
HttpSession: session	Session
ServletContext: application	Application
JspWriter: out	Page
ServletConfig: config	Page
Object: page	Page
▪ Wird normalerweise nicht benutzt.	
Throwable: exception	Page

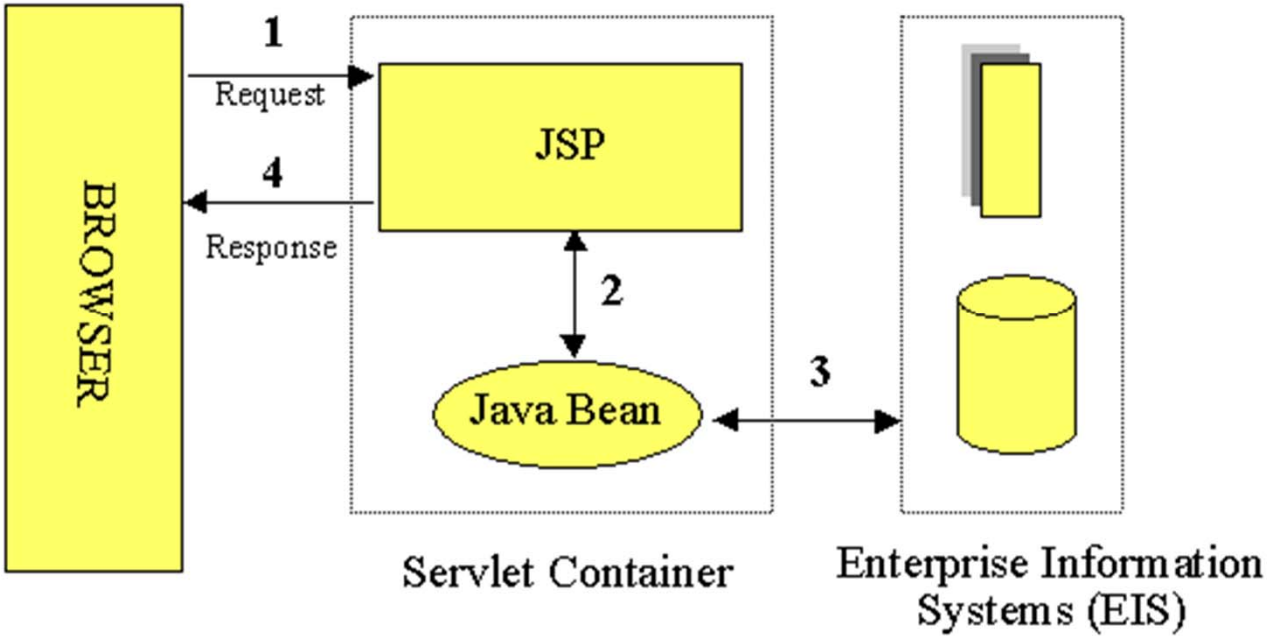
JavaServlets und JavaServer Pages

Model 1 Architecture

Request werden direkt von der JSP-Seite behandelt

JSP-Seite übernimmt Steuerung und Generierung der Antwort

Datenzugriff sollte in Komponenten (JavaBeans) ausgelagert werden



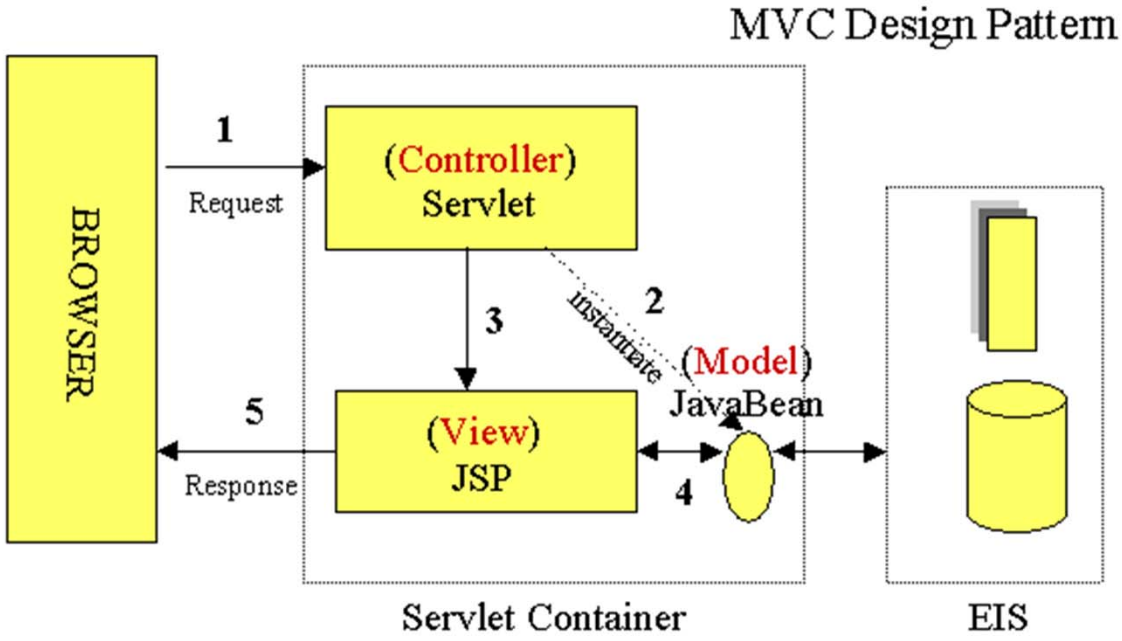
Model 2 Architecture

Serverseitige Realisierung des Model/View/Controller Pattern

Steuerung wird durch ein Servlet übernommen

Generierung des Outputs wird an JSP-Seiten delegiert

Modelkomponenten werden von Servlet an JSP-Seiten weitergegeben



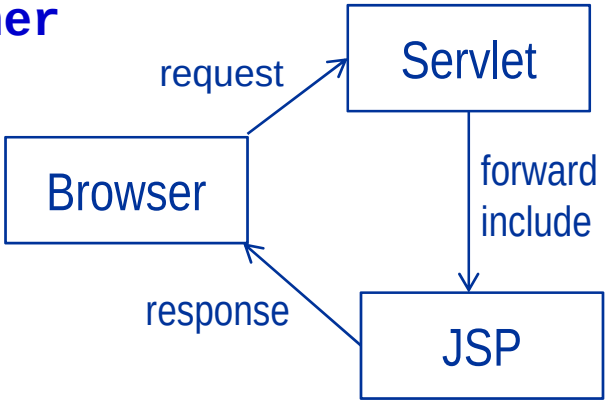
Ziel: Trennung von Logik und Darstellung

Delegation an JSP-Seite

forward / include

Anforderung eines **`javax.servlet.RequestDispatcher`** aus ***`<ServletContext>`***

- **`getRequestDispatcher(String absolutPath)`**
- **`getNamedDispatcher(String name)`**
- **`getRequestDispatcher(String path)`**



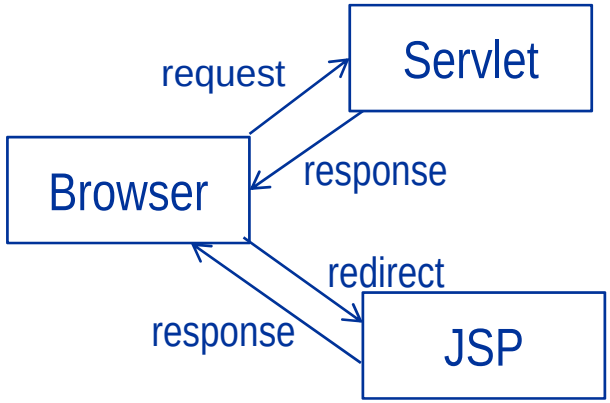
Delegation an JSP-Seite

- **`void include(ServletRequest req, ServletResponse res)`**
  - Inhalt aus JSP einfügen und Kontrolle behalten
- **`void forward(ServletRequest req, ServletResponse res)`**
  - Kontrolle an JSP weitergeben

sendRedirect

Redirect über Response-Objekt

- **`void sendRedirect(String url)`**
  - Redirekt an Seite
  - erfolgt über Browser

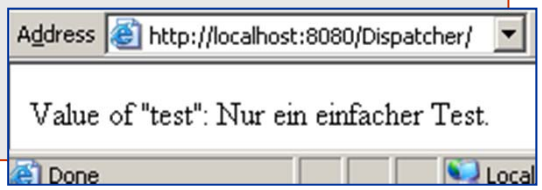


Beispiel RequestDispatcher: forward

```
@WebServlet("/Test")
public class Test extends HttpServlet {

   protected void doGet(HttpServletRequest request,HttpServletResponse response)
         throws ServletException, IOException {
      String test = "Nur ein einfacher Test.";
      request.setAttribute("test", test);

      getServletContext()
                  .getRequestDispatcher("/JSP/test.jsp")
                              .forward(request, response);
   }
}
```

/JSP/test.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1«  … %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
   <title>Insert title here</title>
</head>
<body>
   Value of "test": <%= request.getAttribute("test") %>
</body>
</html>
```



JSP Syntax forward und include

Elemente für Forward und Redirct

- **`<jsp:include>`**
  - Einfügen einer anderen JSP-Seite.
- **`<jsp:forward>`**
  - Kontrolle an eine andere JSP-Seite weitergeben.
- **`<jsp:param>`**
  - Parameterwerte an durch `<jsp:include>` oder `<jsp:forward>` verwendete Seiten weitergeben.

Beispiel forward und redirect

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Validate password</title>
</head>
<body>

…

<%
    if (password.isValid()) {
        response.sendRedirect("ready.jsp");
    } else {
%>
        <jsp:forward page="input.jsp"/>
<%
    }
%>
</body></html>
```

JSP Syntax JavaBeans

JavaBeans-Elemente:

- **`<jsp:useBean>`**
  - JavaBean-Komponente verfügbar machen.

```
<jsp:useBean id="Instanzname" scope="Geltungsbereich"
class="Klassenname"/>
```

- **`<jsp:getProperty>`**
  - Wert einer JavaBean-Eigenschaft abfragen.
- **`<jsp:setProperty>`**
  - Wert von JavaBean-Eigenschaften festlegen.

```
<jsp:setProperty name="Instanzname" property="*" />
```

Übernimmt alle Parameter mit gleichen Namen in Bean

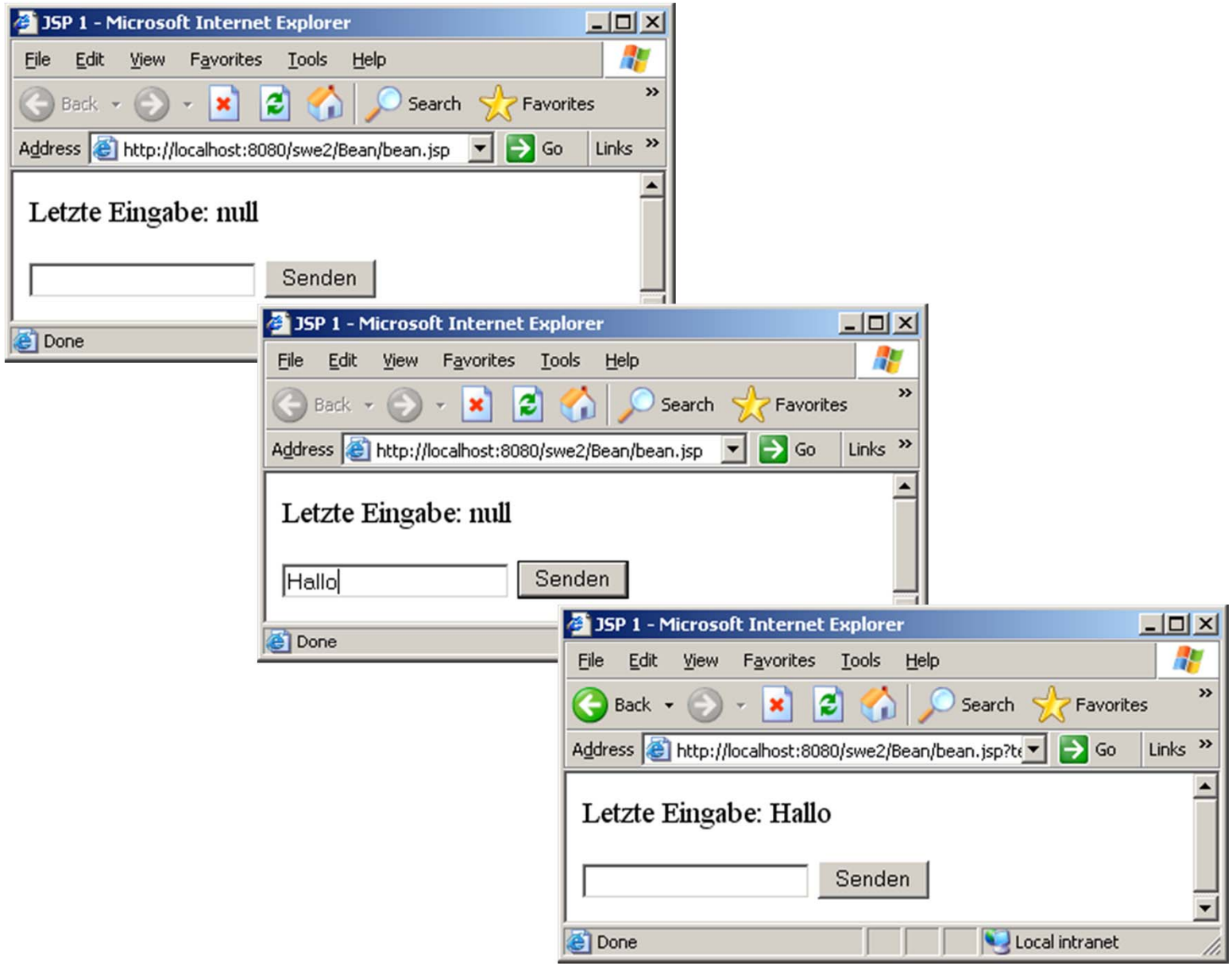
Beispiel JavaBeans: TextBean

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Bean test</title>
</head>
<body>
<jsp:useBean id="textBean" scope="page" class="beans.TextBean"/>
<jsp:setProperty property="*" name="textBean"/>
Last input: <jsp:getProperty property="text" name="textBean"/>
<form action="bean.jsp" method="get">
          <input type="text" name="text"/>
          <input type="submit" name="action" value="Send"/>
</form>
</body>
</html>
```

Setzt alle Properties des Bean mit Werten aus Paramter aus Request wenn gleicher Name

```
package beans;
public class TextBean {
    private String text = "";

    public String getText() {
        return text;
    }
    public void setText(String text) {
        this.text = text;
    }
}
```

Beispiel JavaBeans: TextBean



Beispiel: Password-Validierung

Anwendung zum Abfragen eines gültigen Passwortes (Länge >= 8 Zeichen)

```java
package beans;

public class Password {

    private String password;

    public boolean isInitialized() {
        return password != null;
    }

    public boolean isValid() {
        return password != null && password.length() >= 8;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

}
```

input.jsp:

```
<%@ page language="java" contentType="text/html;
    charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional/
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Password</title>
</head>
<body>
<jsp:useBean id="password" scope="request" class="beans.Password"/>
<% if (password.isInitialized() && !password.isValid()) { %>
    Password is not valid!
<% } %>
Please insert password (> 8 characters):

<form name="text" action="validate.jsp" method="post">

    <input type="text" name =“password"
        value="<%= (password.isInitialized() ? password.getPassword() : "") %>"/>
    <input type="submit" name ="action" value="Send"/>

</form>
</body>
</html>
```

Beispiel: Passwort-Validierung

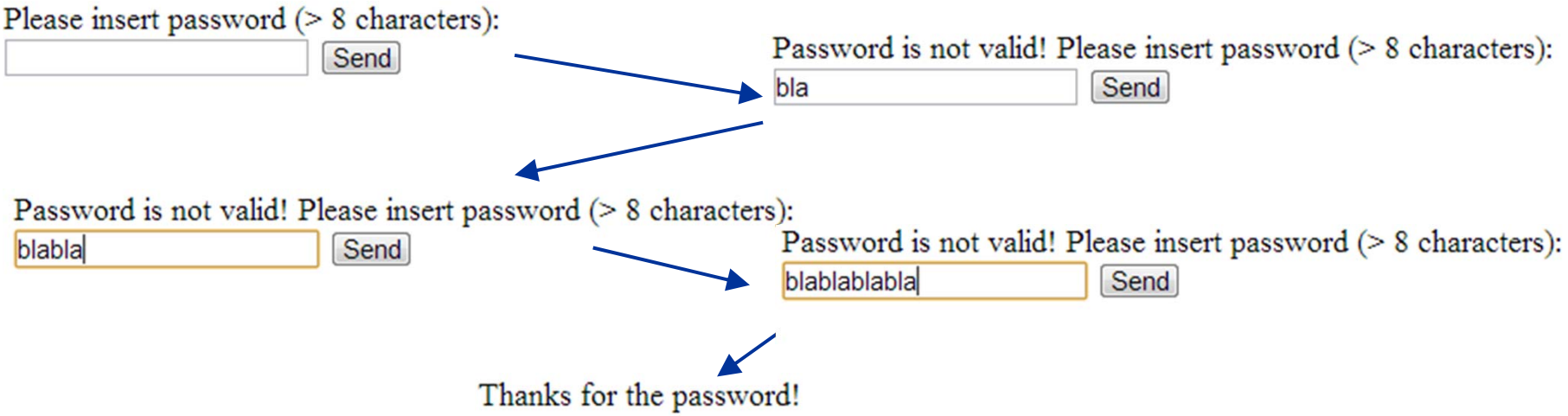
validate.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Validate password</title>
</head>
<body>
<jsp:useBean id="password" scope="request" class="beans.Password"/>
<jsp:setProperty property="password" name="password" param="password"/>

<%
    if (password.isValid()) {
        response.sendRedirect("ready.jsp");
    } else {
%>
        <jsp:forward page="input.jsp"/>
<%
    }
%>
</body></html>
```

Beispiel: Password-Validierung

ready.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%> …
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Ready</title>
</head>
<body>
    Thanks for the password!
</body>
</html>
```



JavaServlets und JavaServer Pages

- Einführung
- Servlets
- Lebenszyklus und Sessions
- Installation
- JavaServer Pages
- Trennung Logik und Darstellung
- Diverses
- Zusammenfassung

Zusammenfassung

Dynamische Webseiten

Sitzungen

JSP ~ ASP/ASP.NET

JSP: Trennung von Aussehen und Implementierung

JSP/Servlet vs. Applet

- Ausführung am Server, Ergebnis: HTML Seite
- Applets: Ausführung am Client, Ergebnis: interaktives Programm

JSP ist abstraktes Servlet

Konfiguration

Download

- Glassfish
  - https://glassfish.dev.java.net/

- Tomcat
  - http://tomcat.apache.org/

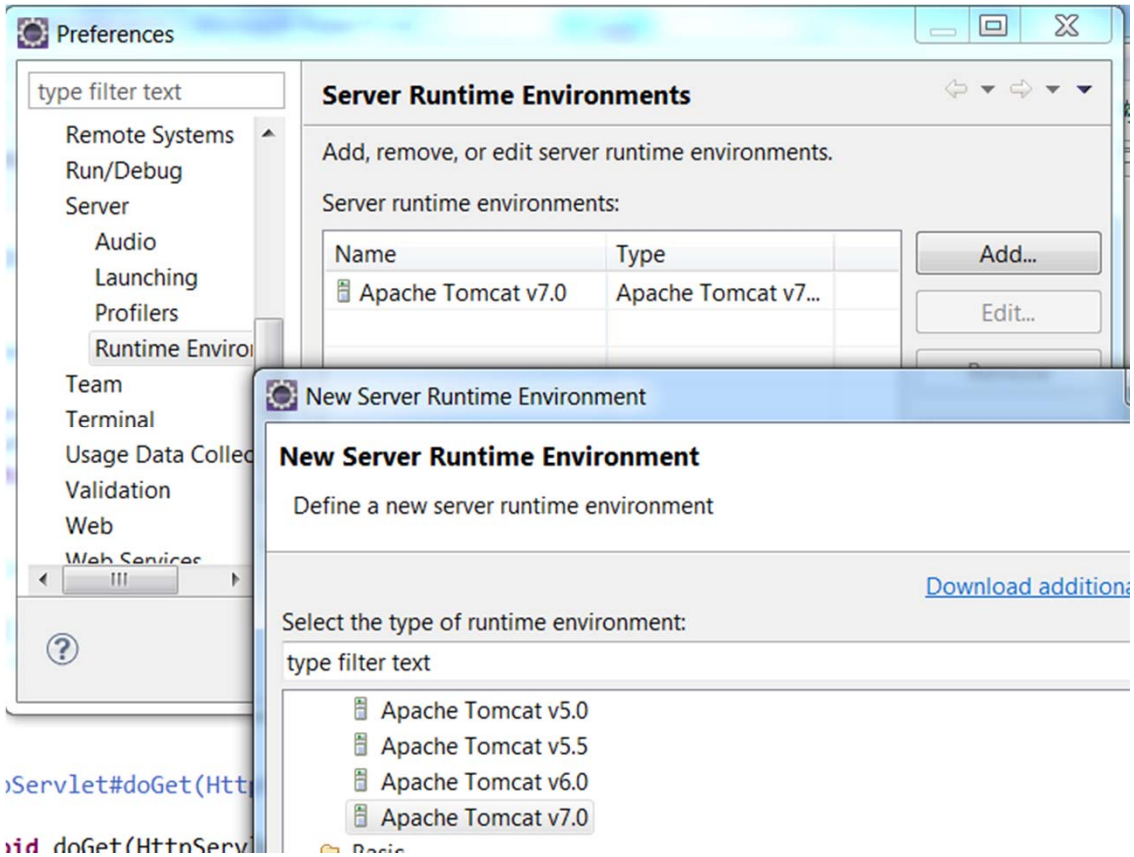
- jetty
  - http://jetty.mortbay.org/

Eclipse JEE

- **Download Eclipse IDE for Java EE Developers**

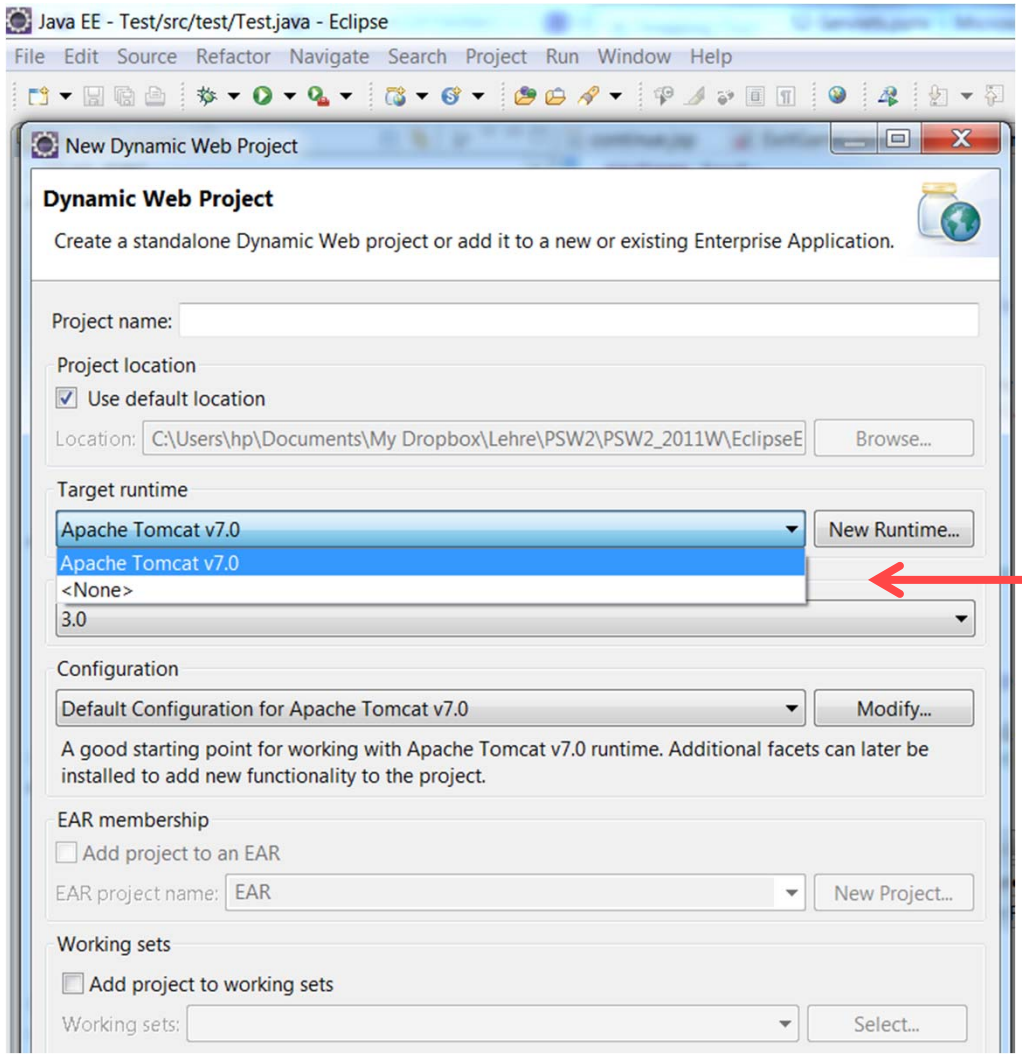
- **Konfiguration des TomCat Servers**

Menü Window -> Preferences -> Server -> Runtime Environment Add…



Eclipse JEE

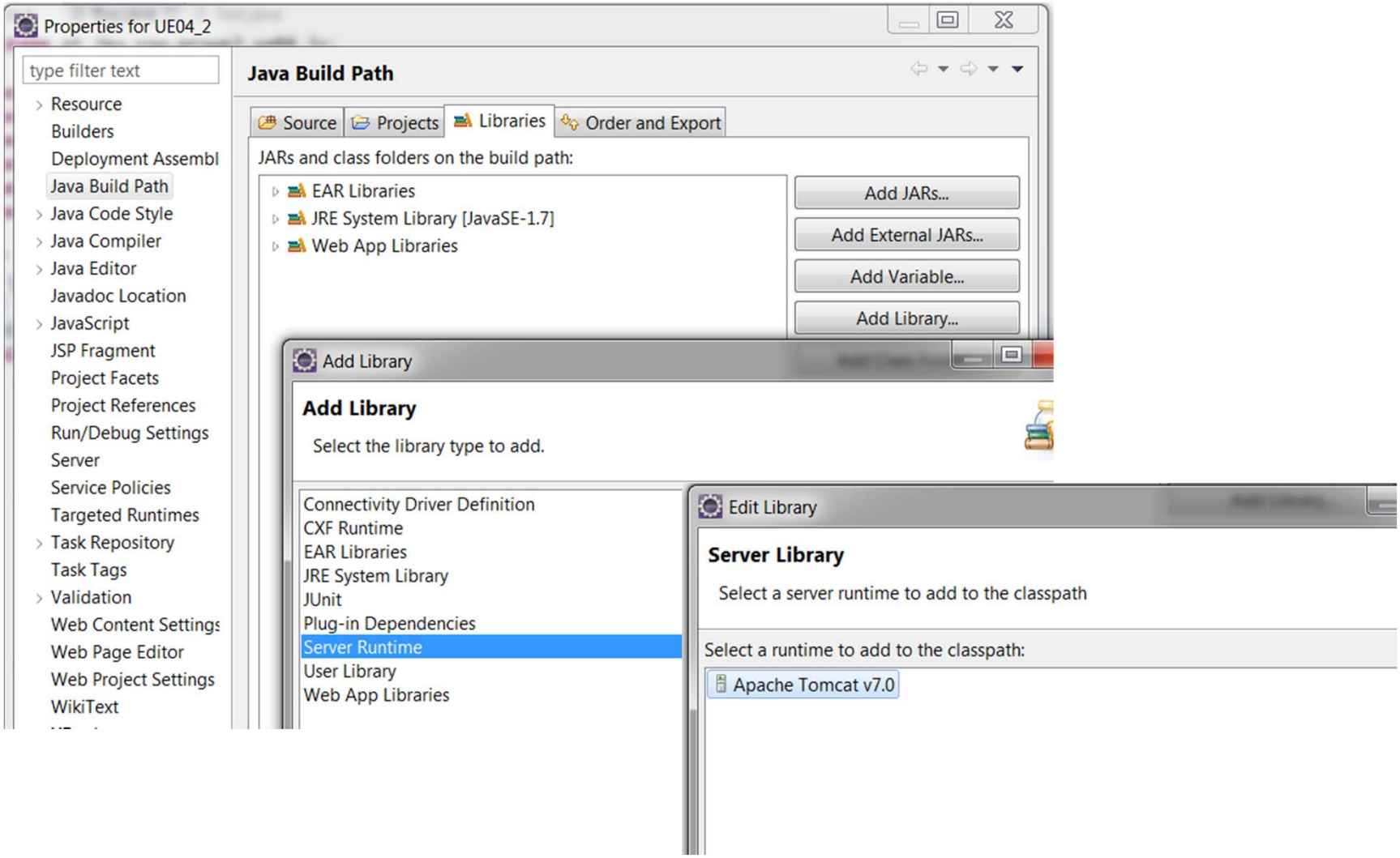
- **Dynamic Web Project erzeugen**



!

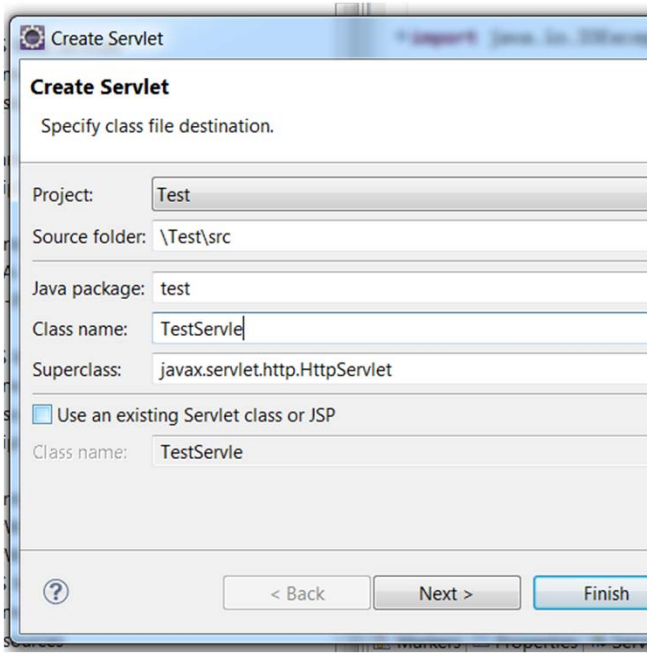
Build-Path

Add Library – Server Runtime – Apache Tomcat



Eclipse JEE

- **Servlet erzeugen**

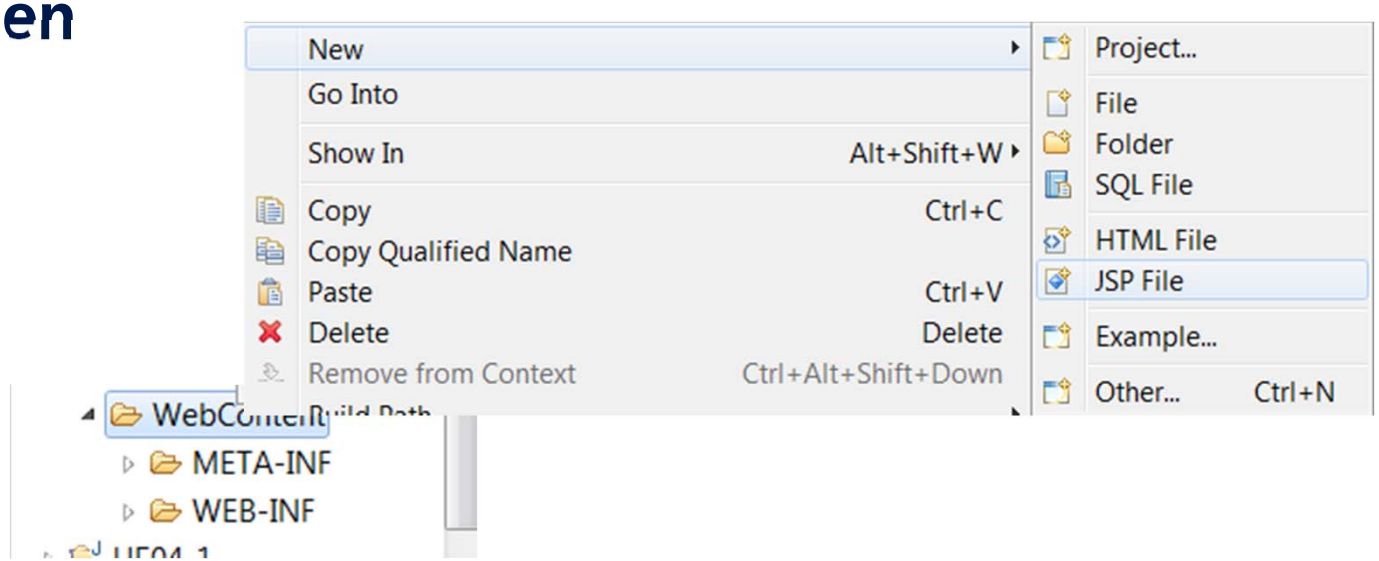


URL für Servlet

```java
package test;

import java.io.IOException;

/**
 * Servlet implementation class TestServlet
 */
@WebServlet("/TestServlet")
public class TestServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public TestServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Ser
        // TODO Auto-generated method stub
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Se
        // TODO Auto-generated method stub
    }
```

Eclipse JEE

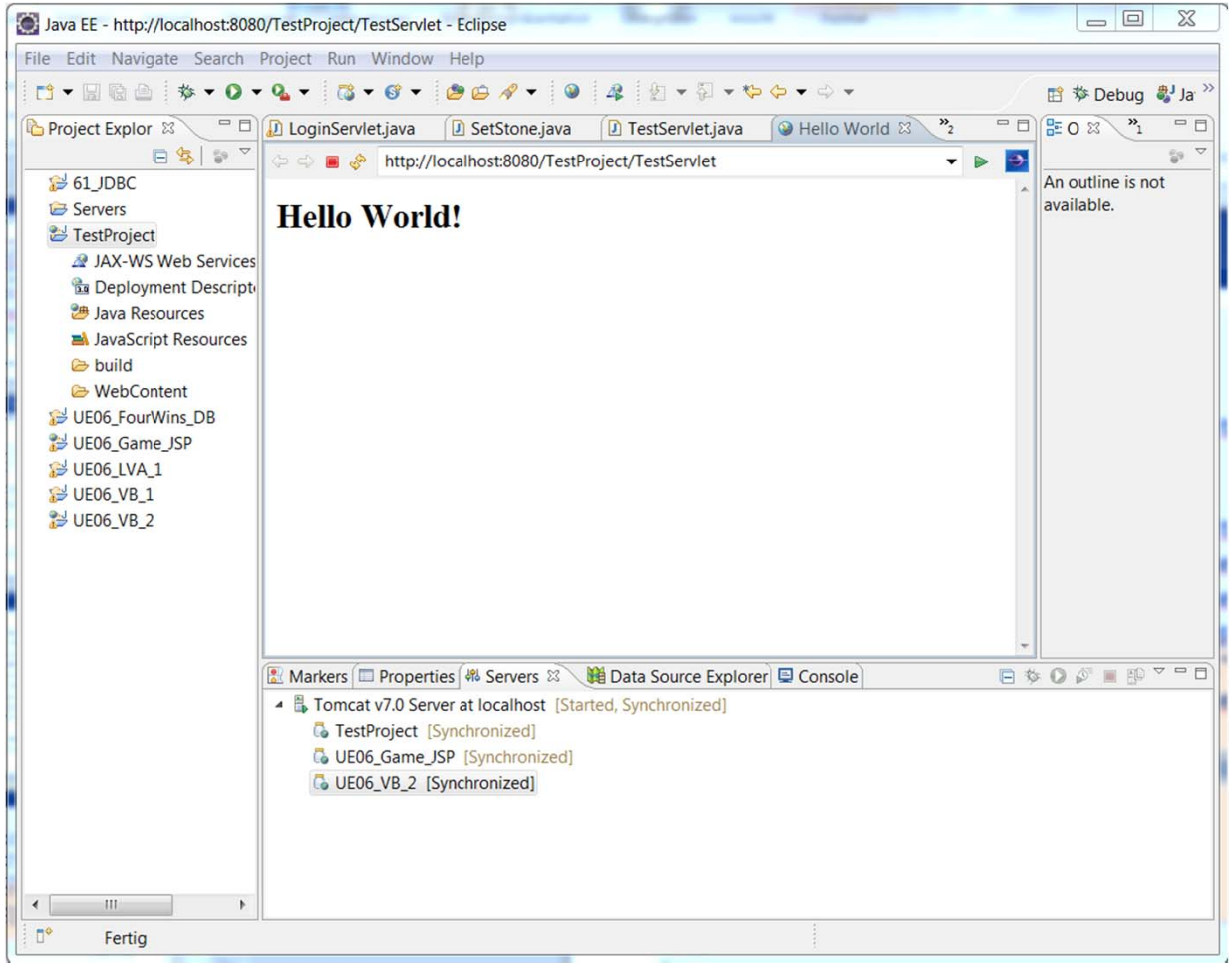
JSP erzeugen




```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

</body>
</html>
```

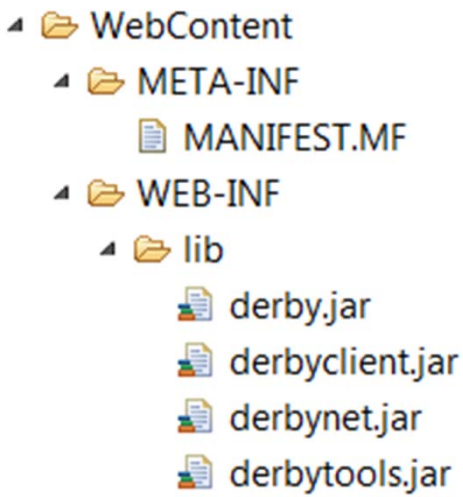
Eclipse JEE

- **Testen von Servlet oder JSP:**
  - Servlet / JSP am Server Tomcat starten



Verwendung von Libraries am Tomcat

Installation in WebContent als Libraries



Beschreibung Installation Tomcat in Eclipse

http://www.se.uni-hannover.de/pages/de:tutorials_javaee_tomcat_eclipse