WBD Zusammenfassung für SEP1

# HTML4

Content-Type: <meta http-equiv='Content-Type' content='text/html;charset=utf-8' />

Meta: <meta name='copyright' content='René Bernhardsgrütter' />

Favicon: <link rel='shortcut icon' href=/favicon.ico' />

CSS: <link rel='stylesheet' type='text/css' href=/style.css' />

# HTML5

Doctype: <!doctype html>

Zeichensatz: <meta charset="UTF-8">

Einfache Tags: header, footer, nav

|  |  |
| --- | --- |
| section | Kapitel oder Abschnitt eines Dokuments |
| article | Eigenständiger Seitenteil, zum Beispiel Blog-Eintrag oder Artikel |
| aside | Block, der nur am Rande mit dem Hauptthema zu tun hat |
| hgroup | Titelbereich einer section |
| <time> | Datum und/oder Uhrzeit |
| <meter> | Massangabe (aus einem Zahlenbereich) |
| <progress> | Fortschrittsangabe (durch Balken visualisiert) |
| <mark> | Text hervorheben |

# CSS

## Seketoren

|  |  |
| --- | --- |
| **Generell** | |
| a.auto.kombi/.auto.kombi | class=‘auto kombi‘ => mehrere Klassen |
| div i | Alle i innerhalb eines div |
| div > p | Alle p die direkt innerhalb eines div liegen (welche dann weiter vererben) |
| div \* b | Alle b, die min. zwei Ebenen innerhalb des div liegen |
| div + p | Alle p, die unmittelbar auf ein div folgen |
| p[title] | Alle p, die ein title-Attribut haben |
| p[title=Einleitung] | Alle p-Elemente, die ein title-Attribut haben, welches den Wert ‚Einleitung‘ besitzt |
| p[title~=Einleitung] | Alle p-Elemente, die ein title-Attribut haben, welches den Wert ‚Einleitung‘ besitzt |
| p[lang|=en] | Alle p-Elemente, dessen lang-Attribut mit ‚en‘ beginnt, gefolgt von einem Bindestrich und dem Rest des Attributwertes |
| **Pseudoklassen** | |
| p:first-line | Erste Zeile des p-Elementes |
| p:first-letter | Erster Buchstabe |
| div p:first-child | Alle p-Elemente, die das erste Kindelement eines divs sind |
| :lang(en) | Alle Elemente, dessen Sprachinhalt als ‚en‘ ausgezeichnet wurde |
| td.Preis:before | Vor dem Inhalt des td der Klasse Preis |
| td.Preis:after | Nach dem Inhalt des td der Klasse Preis. Man kann zusätzlich eine „content:“-Eigenschaft definieren, die Inhalt anzeigt (das geht nur mit before und after): td.Preis:before{ content:“Irgendwas“} Das „Irgendwas“ wird dann „before“ bzw „after“ angezeigt. |

## Counter

Mit dem Counter kann man Inhalt generieren (zählen). Wird von IE nicht unterstützt.

body { counter-reset: ebene1; }

h1:before { content: counter(ebene1) " "; counter-increment: ebene1; }

## Einheiten

|  |  |
| --- | --- |
| Bildschirm | **px** Pixel **em** aktuelle Schriftgrösse **%** prozentual (vergrössern oder verkleinern) |
| Druck | **in** Inch => Nicht verwenden **pt** Punkt (72pt = 1in) **cm** Zentimeter |

## Innen- und Aussenränder, Rahmen

**padding, margin, border zählen nicht zu width (Breite) und height (Höhe).**

Margins zu höheren Ebenen werden gezeichnet. Wenn sich zwei margins derselben Ebene begegnen, überlappen sie sich. Abstand ist dann effektiv so gross, wie das grössere Margin.

|  |  |
| --- | --- |
|  |  |

## Breite, Höhe von Blockelementen

Breite immer so gross wie möglich =>100% (so breit wie das übergeordnete Element), Höhe so klein wie möglich (so gross wie enthaltener Inhalt des Blockelements).

## Positionierung

|  |  |
| --- | --- |
| static | Standardwert |
| relative | Gemessen an Normalposition. Benötigt gleich viel Platz an originalem Ort |
| absolute | Gemessen am Rand des nächst höheren Elements! Benötigt kein Platz an Originalposition. Ignoriert alle Blöcke, die static sind, und berechnet ab dem nächsten Block, der nicht static ist (=relative, fixed, absolute) |
| fixed | Fixiert am Bildschirm, Rest wie bei absolute. |
| float:left; | Element steht links, wird rechts umflossen |
| float:right; | Element steht rechts, wird links umflossen. |
| **import, media, charset** |  |
| @charset "utf-8"; | Zeichensatz wird festgelegt |
| @import url(reset.css); | Importiert eine reset.css |
| @import url(print.css) print, handheld; | Importiert die Datei nur für print und handheld |
| @media print { ... } | Anweisungen nur für print-Geräte Alternative: <link rel="stylesheet" media="print" href="druck.css"> |

# JavaScript

JS definiert das Verhalten der Website. (HTML = Struktur, CSS = Darstellung, JS = Verhalten). Es ist objekt-basiert, jedoch nicht OO. Alles dynamisch. Funktionen sind Objekte.

JS hat ECMA-Standards für herstellerunabhängige Implementierung. Seit 2009 ist **ECMAScript edition 5** aktuell.

Zugriff auf da Filesystem gibt es normalerweise nicht. JSs laufen in einer Sandbox.

Bei Booleschen Vergleichsoperatoren werden Typkonvertierungen durchgeführt mit === und !== kann man jedoch ohne eine Typenkonvertierung fragen, ob etwas identisch oder nicht identisch ist.

## Variablen

var a = „saf“ // Es wird eine Variabl a angelegt. Wenn man kein  
// var voranstellt, istes grobal, mit var lokal.

## Arrays, Objekte

var a = new Array([5]); // a ist ein neues Array mit einer (opt.) Länge 5.

var obj = new Object(); // Objekte sind wie Dict in Python (Name/Wert-Paare)

function sayhi() { alert("hi"); } // Funktionsdeklarateion

obj.sayit = sayhi; // Zuweisung der Funktion in ein Objekt

obj.sayit(); // Aufruf der **Methode** via Objekt

this.irgendwas(); // Ruft innerhalb Obj. Irgendwas() auf

Vordefinierte Objekte, die mit dem Browserfenster und dem angezeigten Inhalt zusammenhängen (enthalten Infos zu Umgebung, welche auch gesetzt werden können):

|  |  |
| --- | --- |
| * Window   + .open() * Document   + .location.href   + .cookie   + .lastModified * Event * History   + .back * Location * Navigator   + .appName   + .appVersion   + .language   + .appCodeName   + .platform   + .userAgent | Vordefinierte Objekte, die überall gehen:   * Array * Date * Function * Math * RegExp * String |

## Datum

Für Datumsverbarbeitung Date()-Objekt. date = new Date() ergibt neues Datumsobjket mit aktueller Zeit. getTime() liefert Unix-Timestamp. Oder temp = new Date(Jahr, Monat, Tag, Std, Min, Sek);. Vorsicht: **Januar ist 0, Dezember ist 11**! Man kann normal damit rechnen.

## Event Handling

*Event Handler:* JavaScript Code, der ausgeführt wird, wenn ein bestimmtes Ereignis (z.B. Mausklick) eintritt. Dies sollte als ausgelagerte Funktion verwendet werden:

function initEvents() { document.getElementById("btn").onclick = alert('Hallo'); }

window.onload = initEvents;

**Einige wichtige Methoden**

onAbort, onChange, onClick, onFocus, **onLoad, onMouseOver, onMousOut,** onReset, onSubmit, onUnload

## Schleifen

for (var i = 0; i < 5; i++) { /\* Code will execute 5 times \*/ }

## document

Mit dem document.X kann man auf Elemente und Eigenschaften des aktuellen Dokuments zugreifen:

* document.cookie Cookie auslesen oder setzen
* document.images, links, forms, … Erlauben Zugriff auf Elemente der Seite

**Zugriff auf Forms**

document.forms[0].elements[0].value = „Hallo Welt“;

## functions

* Haben eigenen Namensraum
* Es gibt keinen Block-Scope
* Funktionszuweisung:
  + Function ggt(x, y) {… return 0;}
* Funktionen können in Objekte gegeben werden
* Function sayhi() {altert ‚hi‘;} Obj.sayit = sayhi; obj.sayhi => hi

## DOM

Hierarchisch aufgebauter Baum. Jeder Knoten (node) hat einen Value, der mit .vaule=‘23‘ belegt werden kann. Attribute des Nodes (z. B. width) wird auch als angehängter node angesehen:

document.getElementById(„header“).width.value = 123;

document.getElementById(„header“).h1.value = „Ich bin ein H1“;

document.getElementById(„header“).h1.width.value = 123

# Serverside-Technologien

## SSI, CGI, Servlets, JSF

**Server-Side-Include:** Es werden mit #include-Tags in HTML-Kommentaren Dateien eingebunden. Einfachste Logik kann ausgeführt werden. Veraltet.

**CGI:** Allgemeine Programmierschnittstelle zw. Webserver und externen Programmen. Häufig mit Shell-Scripts, Perl, Python, aber auch C, PHP verwendet.

**Servlets:** Java-Basistechnologie für alle Java/Web-Sachen.

**JSF:** Komponentenbasiert. Nachfolger von JSP.

## JSP

Veraltet. Wie PHP Einbettung in HTML-Code.

### Markup

|  |  |
| --- | --- |
| <% JavaCode %> | Java-Code wird an entsprechender Stelle ausgeführt |
| <%= Ausdruck %> | Der Ausdruck wird ausgewertet und Resultat ausgegeben. |
| <%! JavaCode %> | Java-Code wird auf Klassenebene (ausserhalb Methode) eingefügt. Z.B. Instanzvariablen. |
| <%-- Kommentar --%> | Kommentar, wird ignoriert. |

### Vordefinierte Variablen

|  |  |
| --- | --- |
| request | HttpServletRequest-Objekt (repräsentiert die Anfrage) |
| response | HttpServletResponse-Objekt (repräsentiert die Antwort) |
| out | PrintWriter-Objekt 🡺response.getWriter() |
| session | Session-Objekt 🡺request.getSession() |
| application | Application-Context 🡺 getServletContext() Applikationsweiter (servletübergreifender) Kontext zum Ablegen von gemeinsamen Daten |
| pageContext | Page-Context, der solange lebt, wie die Generierung der Seite dauert. Gemeinsamer Speicher, wenn die Seite aus mehreren JSP-Dateien aufgebaut wird |
| config | Konfigurationsinfo des Servlets 🡺 getServletConfig() |

### Direktiven

Werden zur Compile-Zeit ausgewertet.

|  |  |
| --- | --- |
| <%@ include file="rel.URL" %> | Fügt Dateiinhalt an der entsprechenden Stelle in die JSP-Seite  ein. |
| <%@ taglib uri="URI" prefix="prefix" /> | Integriert Tag-Library mit spezialisierten XML-Tag-Befehlen. |
| <%@ page xxx="..." %> | kontrolliert die Struktur und Funktionen des generierten Servlets. Wichtigste Inhalte: |
| import="java.util.\*, java.text.\*" |
| content-type="text/html;charset=UTF-8" |
| pageEncoding="UTF-8" |
| session="true" |
| errorPage="MyError.jsp" |
| isErrorPage="true" |

### Actions

Aktionen / Funktionen, die vom JSP-Converter ins Servlet eingebunden werden, jedoch erst zur Laufzeit ausgewertet.

|  |  |
| --- | --- |
| <jsp:forward page="../Login.jsp" /> | Weiterleiten des Requests an eine andere Seite |
| <jsp:include page="../DynamicFooter.jsp" /> | Angegebene Seite wird ausgewertet und das Resultat an dieser Stelle ins Dokument eingefügt. |

### Deployment

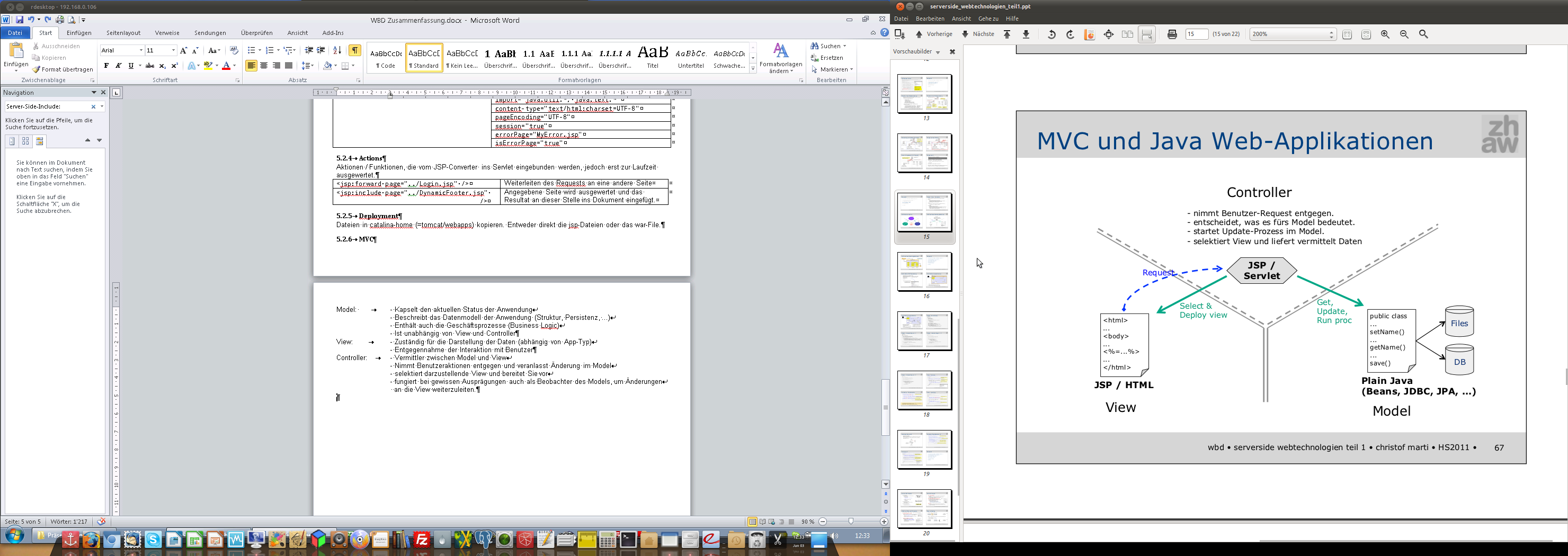
Dateien in catalina-home (=tomcat/webapps) kopieren. Entweder direkt die jsp-Dateien oder das war-File.

### MVC

Model: - Kapselt den aktuellen Status der Anwendung  
- Beschreibt das Datenmodell der Anwendung (Struktur, Persistenz, ...)  
- Enthält auch die Geschäftsprozesse (Business Logic)  
- Ist unabhängig von View und Controller

View: - Zuständig für die Darstellung der Daten (abhängig von App-Typ)  
- Entgegennahme der Interaktion mit Benutzer

Controller: - Vermittler zwischen Model und View  
- Nimmt Benutzeraktionen entgegen und veranlasst Änderung im Model  
- selektiert darzustellende View und bereitet Sie vor  
- fungiert bei gewissen Ausprägungen auch als Beobachter des Models, um Änderungen  
 an die View weiterzuleiten.



### Session

Session erstellen: HttpSession session = request.getSession();

Werte speichern: session.setAttribute("name", wert);

Werte lesen: Objekt wert = session.getAttribute("name");

Session beenden: session.invalidate();

### Scopes

|  |  |
| --- | --- |
| application.setAttribute("name", obj); | Anwendungsweite Attribute, für alle JSPs / Servlets gemeinsam |
| session.setAttribute("name", obj); | Gültig für den aktuellen Benutzer, wird mit Beenden der Session oder schliessen des Browsers gelöscht |
| request.setAttribute("name", obj); | Gültig für die aktuelle Anfrage, mit dem Senden des Antwort, werden diese Attibute gelöscht |
| pageContext.setAttribute("name", obj); | Gültig für die aktuelle Seite, wird nur im Zusamenhang mit Custom-Tag-Libs benötigt. |

### Java Beans

Hält sich an bestimmte Konventionen (z.B. Namensregeln) für Konstruktoren und Methoden.

* Haben zwingend einen Default-Konstruktor.
* Getter: public typ getEigenschaf();
* Setter: public void setEigenschaft(typ);

Lesen: <jsp:getProperty name="entry" property="name" /> (wie entry.getName())

Schreiben: <jsp:setProperty name=„entry" property="name" value="Peter Muster"/>

Alle Properties: <jsp:setProperty name="entry" property="\*" /> 🡺 Schreibt alle Propierties

### Expression Language

**Dot-Notation**: ${entry.name}

**Array-Notation**: ${entry["name"]}