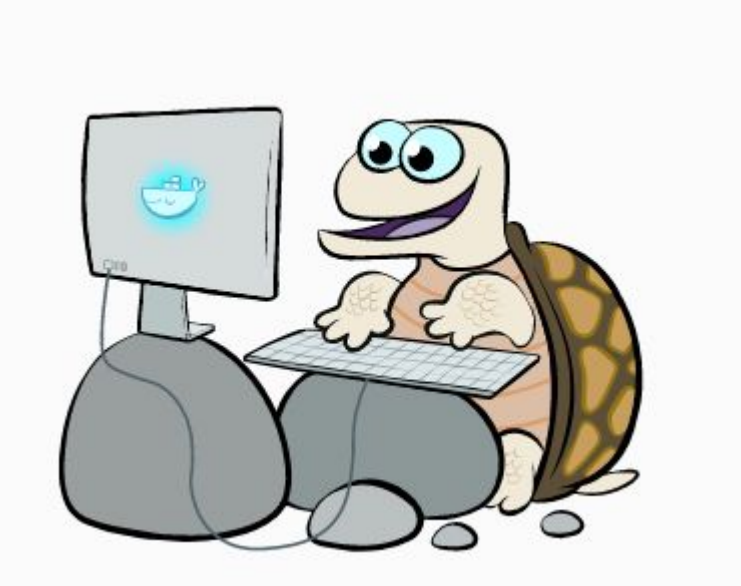
Java 8 & JEE Einführung

Skills

- Java
- maven

Ziel der Schulung : JEE Basics

- JEE - Basis wissen
- JEE - Überblick

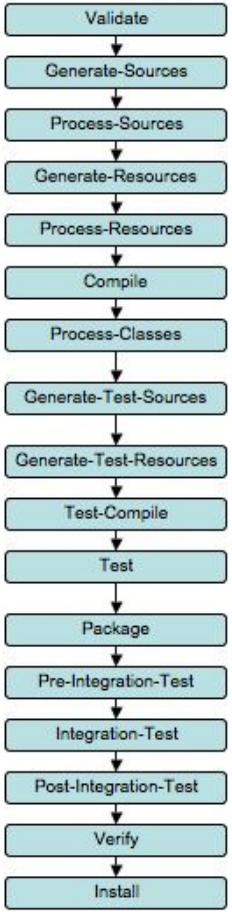


Maven - Basics - Auffrischung

Maven

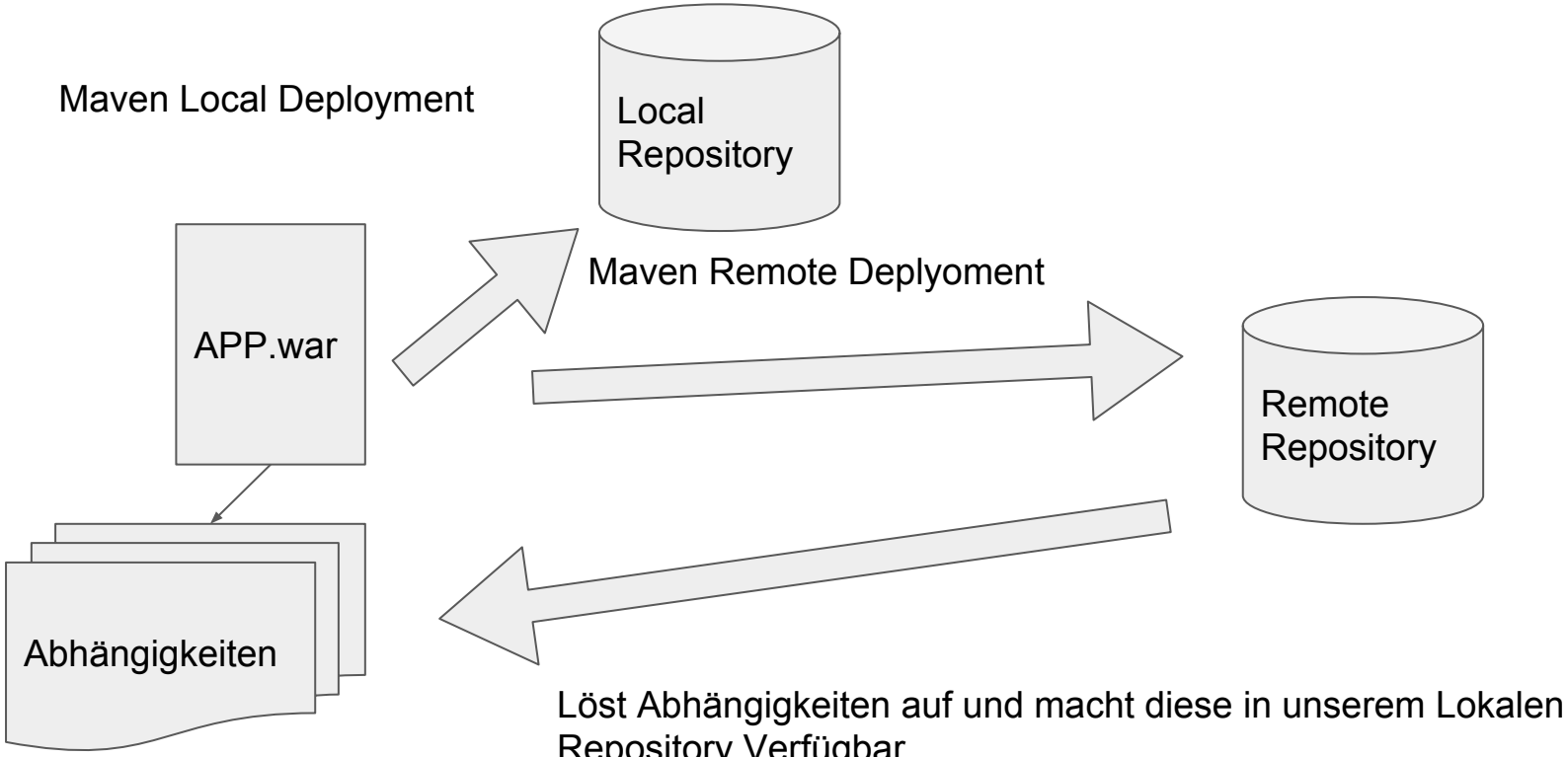
- Maven ist ein Build-Management-Tool wie Ant oder Gradle
- basiert auf Java
- Maven versucht den Grundgedanken “Konvention vor Konfiguration” konsequent im gesamten Entwicklungszyklus abzubilden.
- Bleibt man nah am Maven Standard so braucht man sich um die Aufgaben des Build Managements kaum zu kümmern.
- Offt Reichen Wenige Konfigurationseinstellungen

Der Standard-Lebenszyklus



Quelle: Eclipse.org

Maven Repository - Dependency Management



Maven Befehl zum Bauen & ihr Unterschied

mvn clean install

-> clean => lösche das “target”-Verzeichnis

-> install => führe bis zur Installationsphase den Maven Build durch

mvn clean package

-> clean => lösche “target”-Verzeichnis

-> install => führe bis zur “package”-Phase den Maven Build durch

-> Artifakt wird nicht ins Maven lokale Repository deployed (.m2 im Homeverzeichnis)

Java 8 Features

Java 8 - Lambda-Funktion

- Eine anonyme Funktion
- auch Lambda-Funktion genannt
- wird nicht per Namen sonder per Referenz angesprochen
- Geschlossener funktionaler Kontext, auch Closure genannt (eigener Speicherbereich u. Zustand)
- return bei Einzeilern nicht notwendig
- Lambda-Funktionen können ihre Fähigkeiten vor allem im dynamischen und nicht streng-typisierten Programmiersprachen entfalten

Java8 - Lambda-Funktion - Beispiel

```
public class ClosureAndLambda {

 interface Math {
  int func(int number1, int number2);
  int func(int... numbers); //wie int[] numbers
 }

 public static void main(String...args) throws Exception {
  Math add = (int number1, int number2) -> number1 + number2;
  Math mul = (int number1, int number2) -> number1 * number2;
  Math customCalc = (int number1, int number2) -> {
   return 2 * number1 + 3 * number2;
  };
  System.out.println(customCalc.func(1, 2));
 }

}
```

Java 8: Lambda expressions

Beispiel in Java 7:

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, new Comparator<String>() {
    @Override
    public int compare(String a, String b) {
        return b.compareTo(a);
    }
});
```

Java 8: Lambda expressions

Ab Java 8:

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, (String a, String b) -> {
    return b.compareTo(a);
});
```

Java 8: Lambda expressions

Es geht noch kürzer:

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, (String a, String b) -> b.compareTo(a));
```

Java 8: Lambda expressions

Und noch kürzer:

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, new Comparator<String>() {
    @Override
    public int compare(String a, String b) {
        return b.compareTo(a);
    }
});
```

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, (a, b) -> b.compareTo(a));
```

Java 8: Method references

Ab Java 8 können auch Methoden referenziert werden:

```
Function<String, Integer> stringToInt = Integer::valueOf;

Integer onetwothree = stringToInt.apply( t: "123");

Function<String, Integer> stringToIntSquared = stringToInt.andThen((i) -> i * i);

Integer bigNumer = stringToIntSquared.apply( t: "123");
```

Java 8: Supplier und Consumer

Supplier:

```
Supplier<Person> personFactory1 = PersonImpl::new;
Person p1 = personFactory1.get();

Supplier<Person> personFactory2 = () -> new PersonImpl();
Person p2 = personFactory2.get();

Supplier<Integer> intGetter = () -> new Random().nextInt();
Integer randomInt = intGetter.get();
```

Consumer:

```
Consumer<Person> helloMethod = Person::sayHello;
helloMethod.accept(p1);

Consumer<String> sysout = System.out::println;
sysout.accept( t: "hello there");

Consumer<Integer> ageSetter = p2::setAge;
ageSetter.accept( t: 18);
```

Java 8: Streams

Kann auf beliebigen Collections ausgeführt werden:

```
List<String> stringCollection = new ArrayList<>();
stringCollection.add("ddd2");
stringCollection.add("aaa2");
stringCollection.add("bbb1");
stringCollection.add("aaa1");
stringCollection.add("bbb3");
stringCollection.add("ccc");
stringCollection.add("bbb2");
stringCollection.add("ddd1");

stringCollection
        .stream()
        .filter((s) -> s.startsWith("a"))
        .forEach(System.out::println);
```

Java 8: Streams

Es kann sortiert und “gemapped” werden:

```
stringCollection
        .stream()
        .map(String::toUpperCase)
        .sorted((a, b) -> b.compareTo(a))
        .forEach(System.out::println);

List<Integer> intList = stringCollection
        .stream()
        .map((s) -> Integer.valueOf(s) + 10)
        .collect(Collectors.toList());
```

Java 8: Streams

Es kann gesucht werden:

```
boolean anyStartsWithA =
        stringCollection
                .stream()
                .anyMatch((s) -> s.startsWith("a"));

System.out.println(anyStartsWithA);      // true

boolean allStartsWithA =
        stringCollection
                .stream()
                .allMatch((s) -> s.startsWith("a"));

System.out.println(allStartsWithA);      // false

boolean noneStartsWithZ =
        stringCollection
                .stream()
                .noneMatch((s) -> s.startsWith("z"));

System.out.println(noneStartsWithZ);      // true
```

Java 8: Streams

Es kann auch gezählt werden:

```
long startsWithB =
        stringCollection
                .stream()
                .filter((s) -> s.startsWith("b"))
                .count();

System.out.println(startsWithB);    // 3
```

Java 8: Streams

Und zuletzt können die Elemente kombiniert werden:

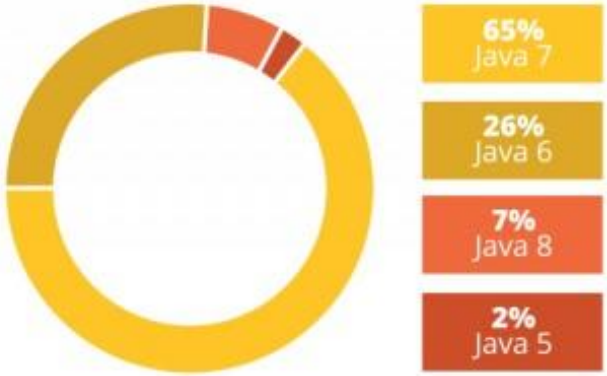
```
List<String> stringCollection = new ArrayList<>();
stringCollection.add("ddd2");
stringCollection.add("aaa2");
stringCollection.add("bbb1");
stringCollection.add("aaa1");
stringCollection.add("bbb3");
stringCollection.add("ccc");
stringCollection.add("bbb2");
stringCollection.add("ddd1");

Optional<String> reduced =
        stringCollection
                .stream()
                .sorted()
                .reduce((s1, s2) -> s1 + "#" + s2);

reduced.ifPresent(System.out::println);
// "aaa1#aaa2#bbb1#bbb2#bbb3#ccc#ddd1#ddd2"
```

Wie viele % Entwickler nutzen JEE ?

WHO IS USING JAVA EE?

Java EE versions in use*

Java SE versions in use



* *The results were normalized to exclude non-users*

REBELLABS

Why ?

Standardisierung

bedeutet im eigentlichen Wortsinn eine Vereinheitlichung von Maßen, Typen, Verfahrensweisen, Strukturen oder anderem. Ziel ist die Schaffung gemeinsamer Standards respektive Parameter (beispielsweise bei Werkzeugen, Produktions- oder Softwarekomponenten)

Skalierbarkeit

versteht man die Fähigkeit eines Systems, Netzwerk oder Prozesses zur Größenveränderung. Meist wird dabei die Fähigkeit des Systems zum Wachstum bezeichnet.

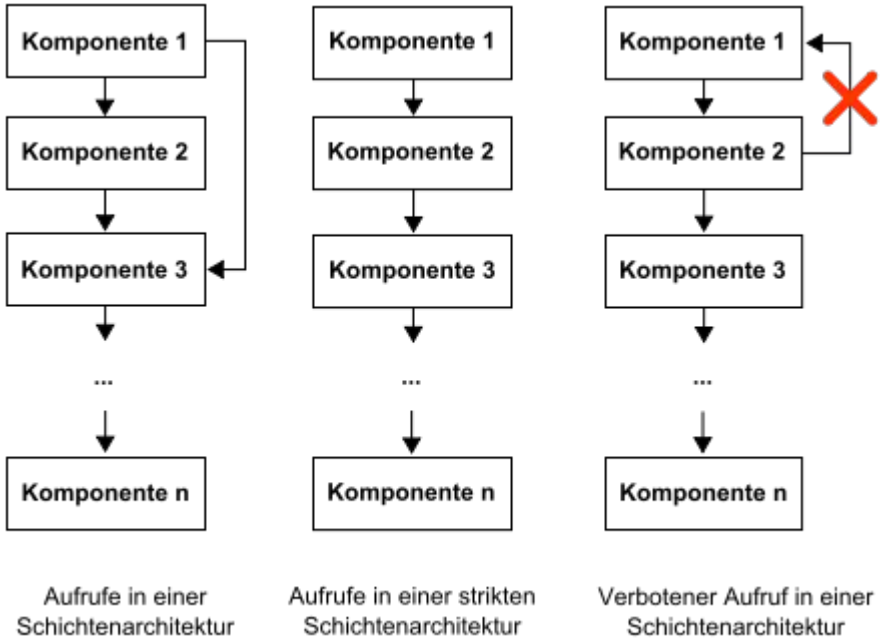
Interoperabilität

bezeichnet man die Fähigkeit zur Zusammenarbeit von verschiedenen Systemen, Techniken oder Organisationen

aus Wikipedia

Grundlagen

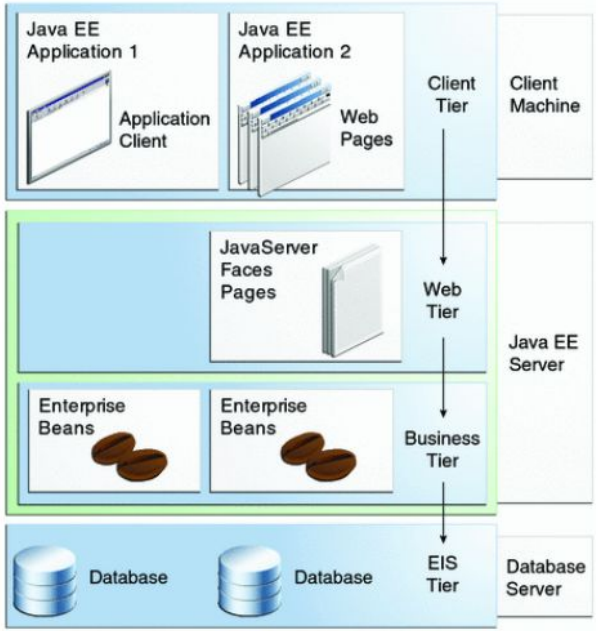
N-Tier-Architektur - oder auch Schichtenarchitektur



Quelle: wikipedia.de

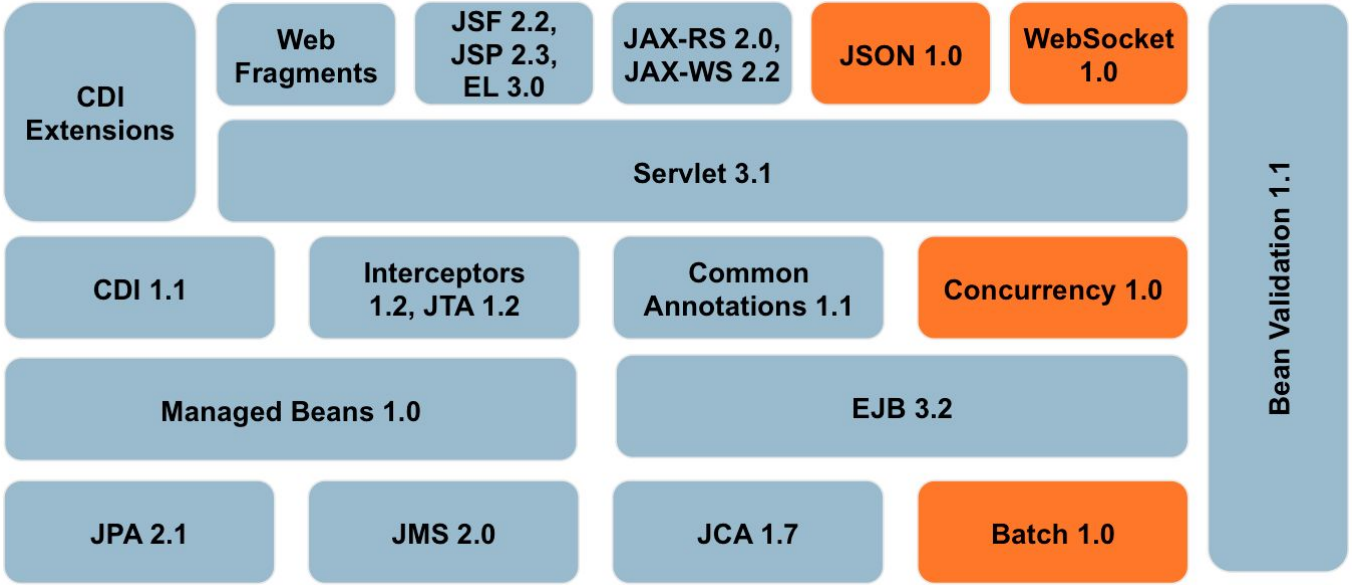
Überführt auf JEE

- Presentation Tier (Web Tier)
    - JSF Pages (Java Server Faces)
    - JSF Backing Beans
- Business Tier
    - Enterprise Beans
    - JTA (Java Transaction API)
- Persistence Tier
    - Entity Beans
    - JPA (Java Persistence API)



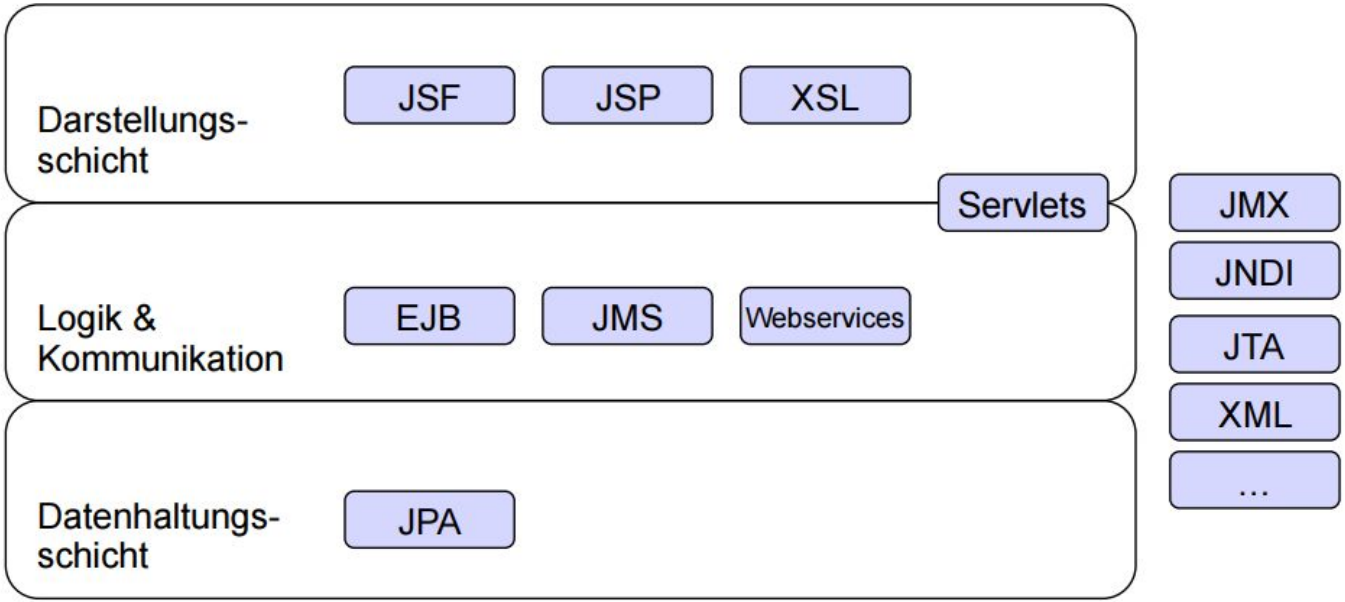
Quelle : docs.oracle.com

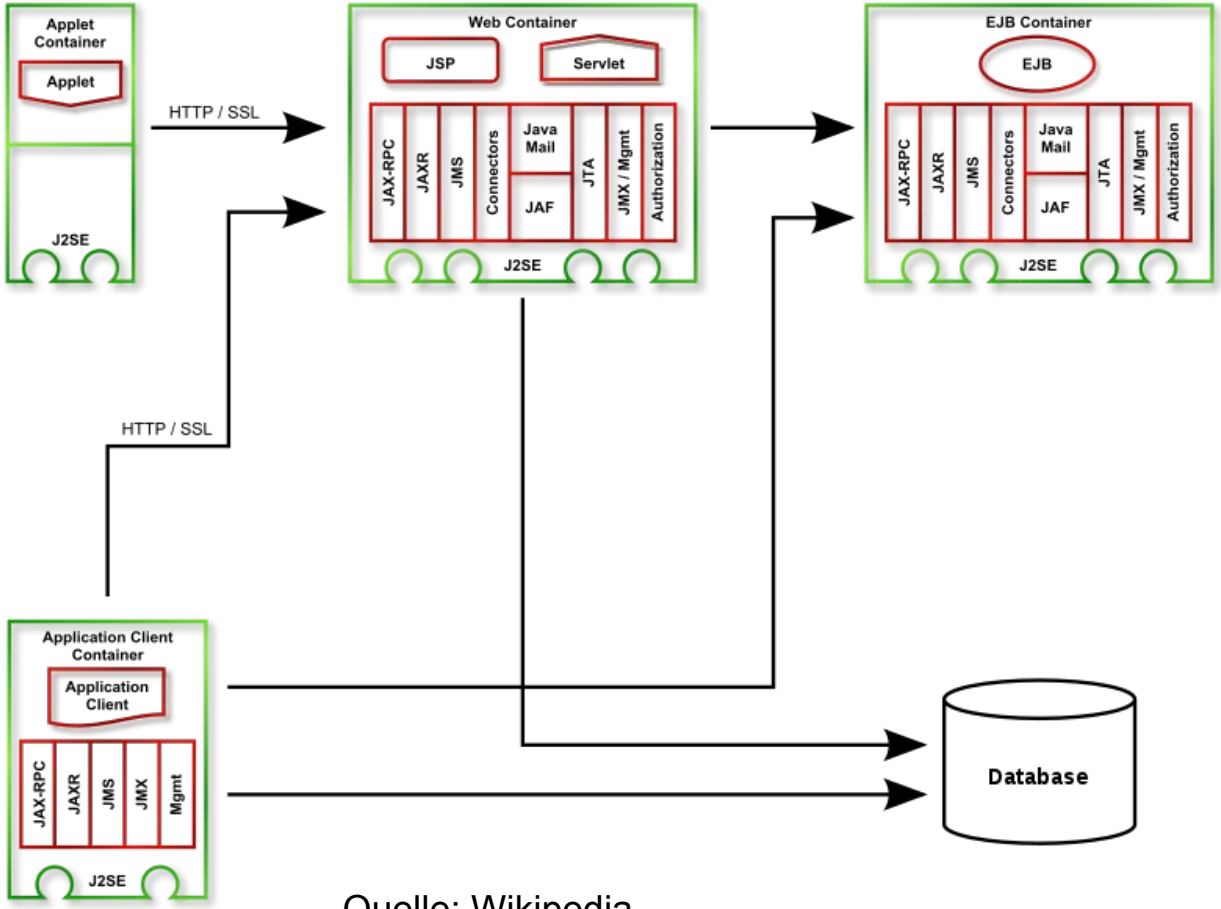
JEE Komponenten



Quelle: Sahet.NET

Einordnung der APIs



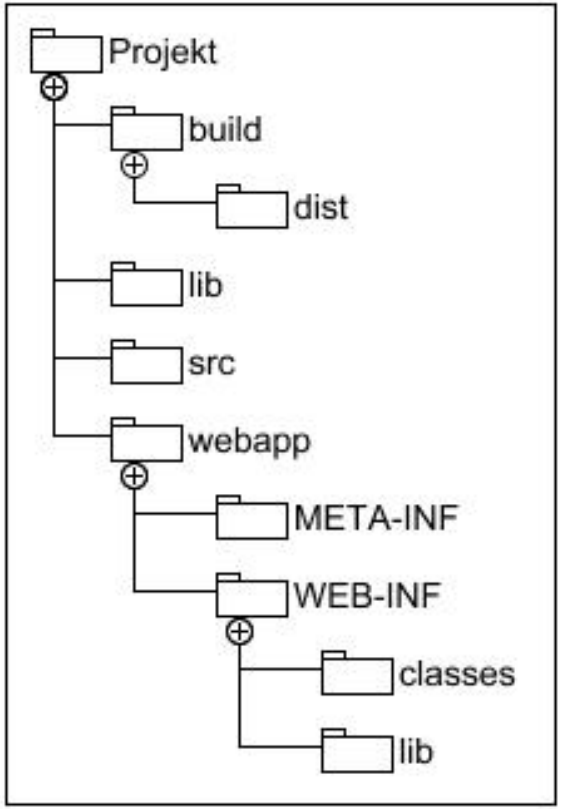


Quelle: Wikipedia

Java Application Server

- Apache Tomcat, Geronimo, JBoss jetzt Wildfly, Websphere, WebLogic oder Glassfish

- Hier werden verschiedene Software Arten Deployed, die Häufigste ist das WAR file **W**eb **A**pplication **R**esource oder auch **W**eb application **AR**chive



Quelle:

http://localhost:8080/hello-world-war-1.0.0/

Java Web Applikationen

Servlets, Java Server Pages und Java Server Faces

Was ist ein Servlet

Als Servlets bezeichnet man Java-Klassen, deren Instanzen innerhalb eines Webservers Anfragen von Clients entgegennehmen und beantworten.

Interface HttpServlet

https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html

Wichtige Methoden :

init() - wird aufgerufen beim Hochfahren des Servlets

destroy() - wird beim Herunterfahren des Servlets aufgerufen

Verschiedenen Service Methoden

diese Laufen jeweils pro HTTP Request einmal.

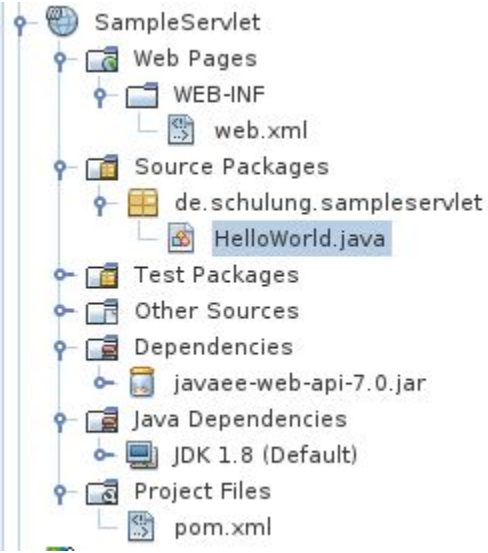
Jede diese Methoden haben als Übergabe Parameter **request** and **response** Objekte in ihrer Signatur was uns Erlaubt auf diese Zuzugreifen.

doGet, für HTTP GET requests

doPost, für HTTP POST requests

doPut, für HTTP PUT requests

doDelete, für HTTP DELETE requests

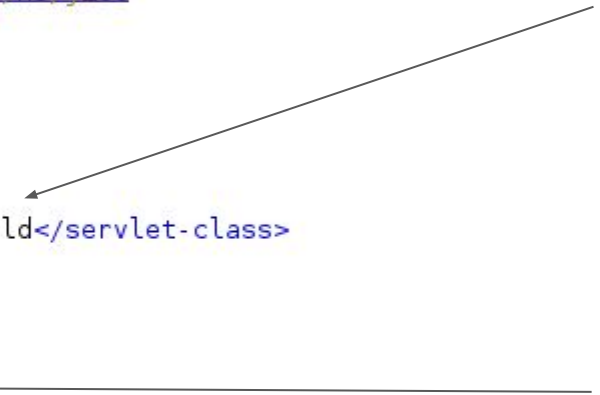



```java
package de.schulung.sampleservlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author chdi
 */
public class HelloWorld extends HttpServlet {

    private String message;

    public void init() throws ServletException {
        // Do required initialization
        message = "Hello World";
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html");

        // Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message + "</h1>");
    }

    public void destroy() {
        // do nothing.
    }
}
```

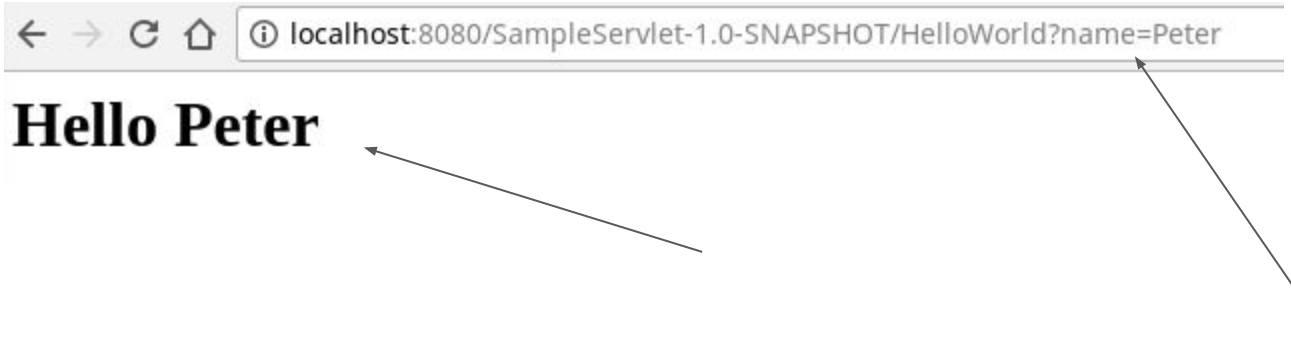
Example Servlet

Web.xml

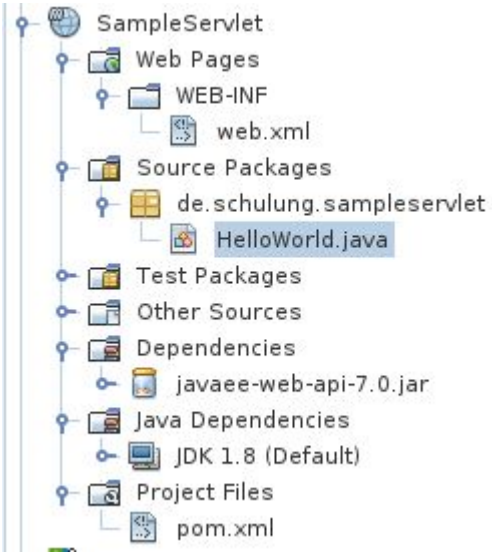
```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
5   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
6
7   <servlet>
8       <servlet-name>HelloWorld</servlet-name>
9       <servlet-class>de.schulung.sampleservlet.HelloWorld</servlet-class>
10  </servlet>
11
12  <servlet-mapping>
13      <servlet-name>HelloWorld</servlet-name>
14      <url-pattern>/HelloWorld</url-pattern>
15  </servlet-mapping>
16
17  </web-app>
```



Aufgabe : Erweitern sie das Servlet



Lösung:



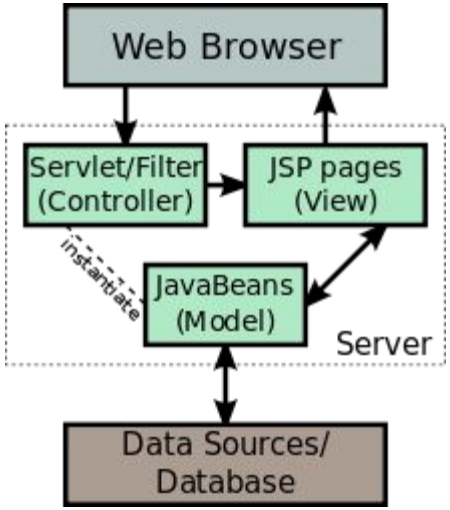

```java
package de.schulung.sampleservlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author chdi
 */
public class HelloWorld extends HttpServlet {

    private String message;

    public void init() throws ServletException {
        // Do required initialization
        message = "Hello";
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html");


        String name = request.getParameter("name");


        // Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message +" "+ name + "</h1> ");
    }

    public void destroy() {
        // do nothing.
    }
}
```

JSP - JAVA SERVER PAGES

JSP ist die Antwort auf das populäre ASP - Active Server Pages von Mircosoft.

JSP bietet uns einen einfachen Weg dynamische Web Inhalte zu erstellen, es erlaubt uns **STATISCHEN** HTML Code und **DYNAMISCH** generierten HTML Code zu mischen.

Somit können wir unsere Business-Logic von der Ansicht separieren.



Quelle: Wikipedia

Die Vorteile von JSP sind:

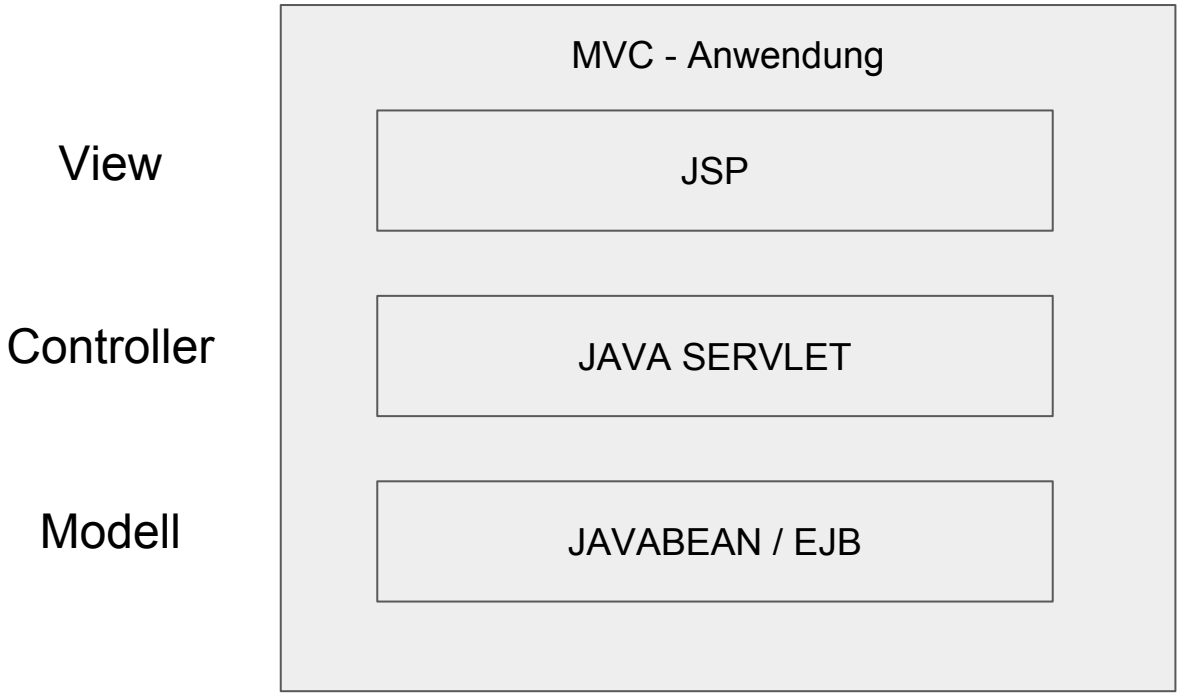
- “Trennung” von statischen und dynamischen Inhalten
- Wiederverwendung von Komponenten und Tag-Bibliotheken
- Java Power und Portabilität

JSPs werden intern in Java Servlets kompiliert

Das heißt :

- alles, was mit JSPs gemacht werden kann, kann auch mit Java-Servlets erreicht werden.
- Es ist jedoch wichtig anzumerken, dass Servlets und JSPs komplementäre Technologien sind und NICHT gegeneinander ausgetauscht werden.
- Servlet kann als "HTML innerhalb von Java" angesehen werden, was besser für die Implementierung von Geschäftslogik ist - da es Java dominiert. JSP hingegen ist "Java inside HTML", das für die Erstellung von Präsentationen überlegen ist - da es HTML-dominant ist.

Beispiel : Model-View-Control (MVC Pattern)

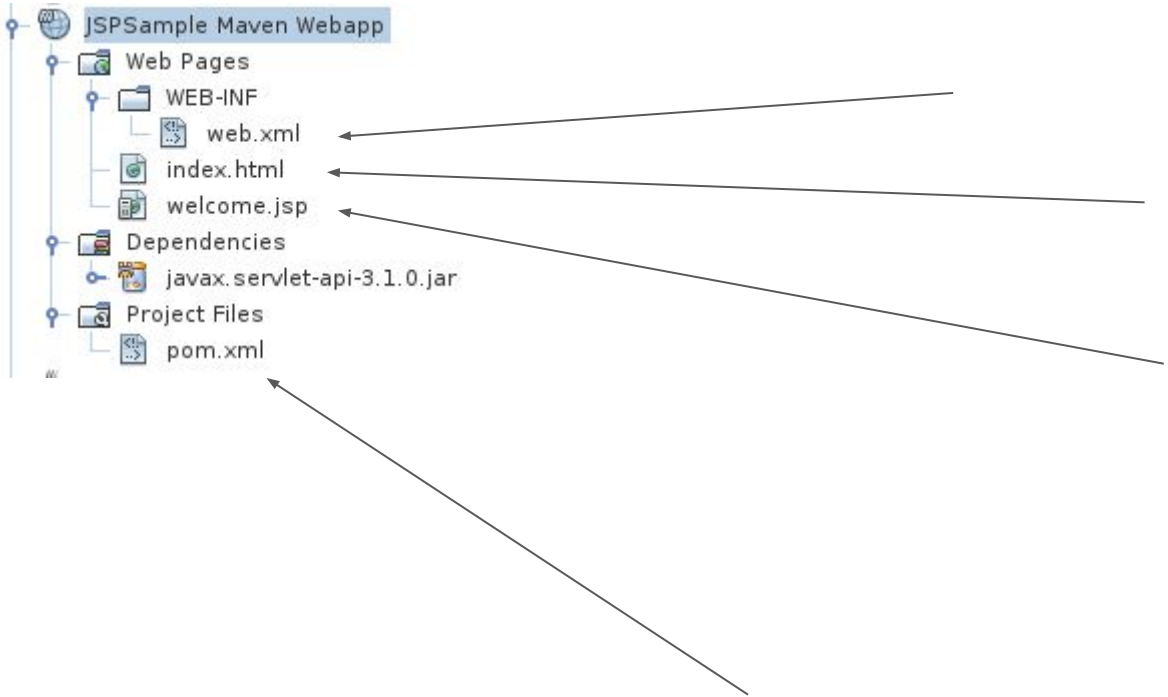


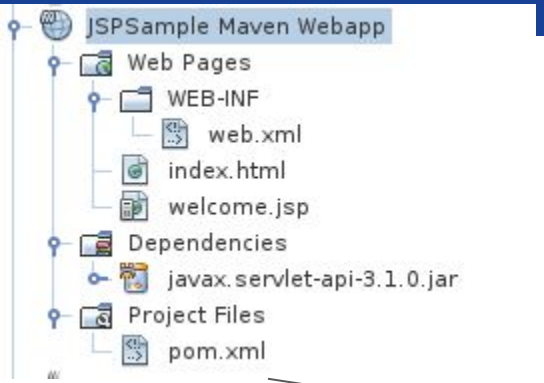
JSP Beispiel Application

Welcome Max Mustermann

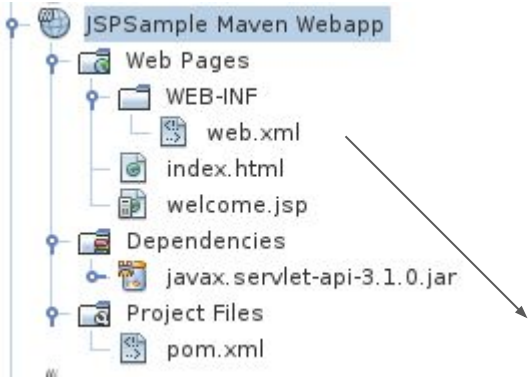
```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Projekt Struktur

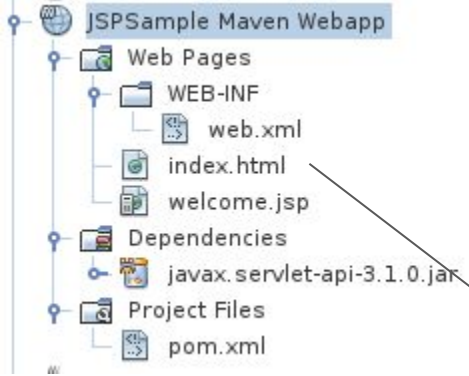




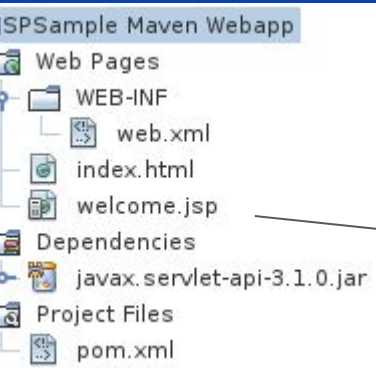

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://ww
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.ap
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.home.jsp</groupId>
  <artifactId>JSPSample</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>JSPSample Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <properties>
  <servlet.version>3.1.0</servlet.version>
  <maven.compiler.plugin.version>3.5.1</maven.compiler.plugin.version>

  </properties>

  <dependencies>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>${servlet.version}</version>
    </dependency>
  </dependencies>


  <build>
    <finalName>JSPSample</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>${maven.compiler.plugin.version}</version>
        <configuration>
          <!-- Java version-->
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```




```
<!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>
  <display-name>JSP Sample Example</display-name>
</web-app>
```




```
<html>
<body>
<h2>Welcome</h2>

<form action="welcome.jsp" method="get">

Enter username: <input type="text" name="uname" />
<br/>
<br/>
Enter numbr for mulitplication tables: <input type="text" name="number" />
<br/>
<br/>
<input type="submit"/>
<br/>
<br/>

</body>
</html>
```




```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>Multiplication tables</title>
</head>
<body>

<h3>Welcome <%=request.getParameter("uname") %></h3>
<pre>
<%
String value = request.getParameter("number");
try
{
        Integer number = Integer.parseInt(value);
        for(int i=1;i<=10;i++)
        {
            out.println(number + " * " + i +" = " + (number*i)  );
        }
}
catch(NumberFormatException e)
{
        out.println("Invalid number");
}
%>
<pre>
<br/>
<br/>


</body>
</html>
```

Modularer Aufbau JSP (Composite-Pattern)

The include directive
(translation time)

```
<%@ include file="header.html" %>
```

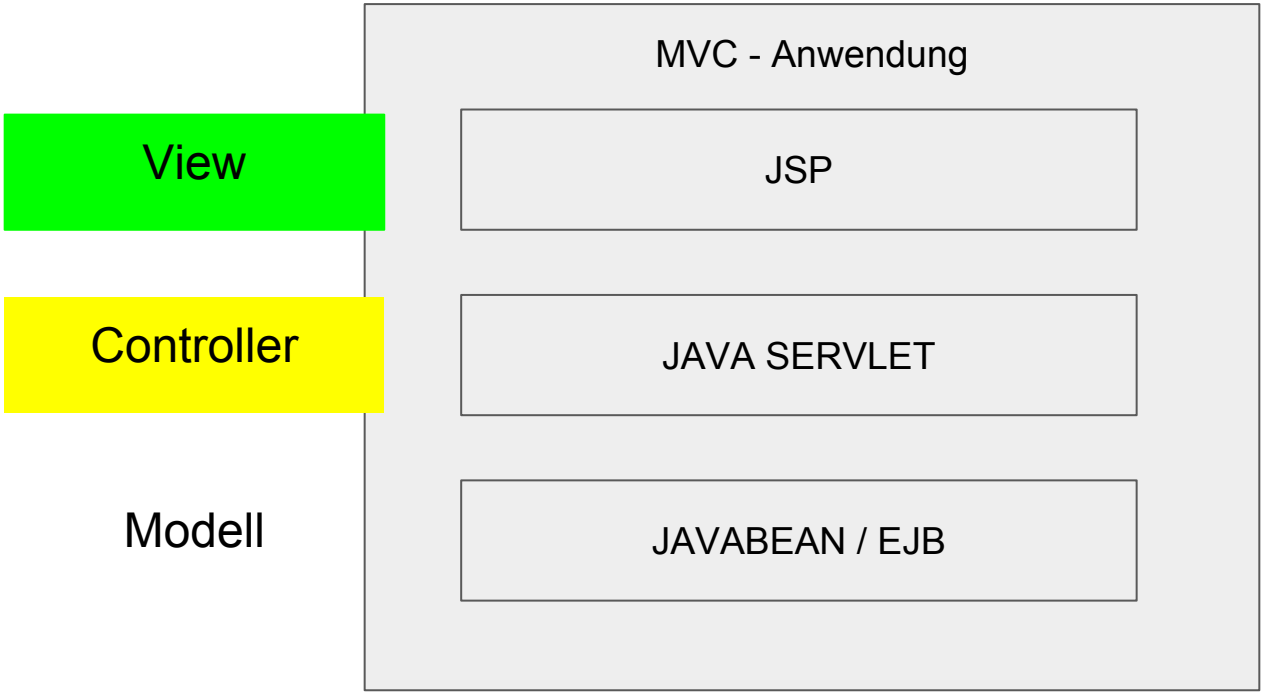
The <jsp:include>
(request time & inside)

```
<jsp:include page="header.jsp" />
```

The <c:import> JSTL tag
(request time & inside, outside)

```
<c:import url="http://www.example.com/foo/bar.html" />
```

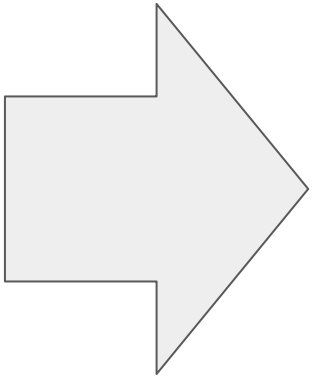
JSP - Servlets im Zusammenspiel



Welcome Max Mustermann

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

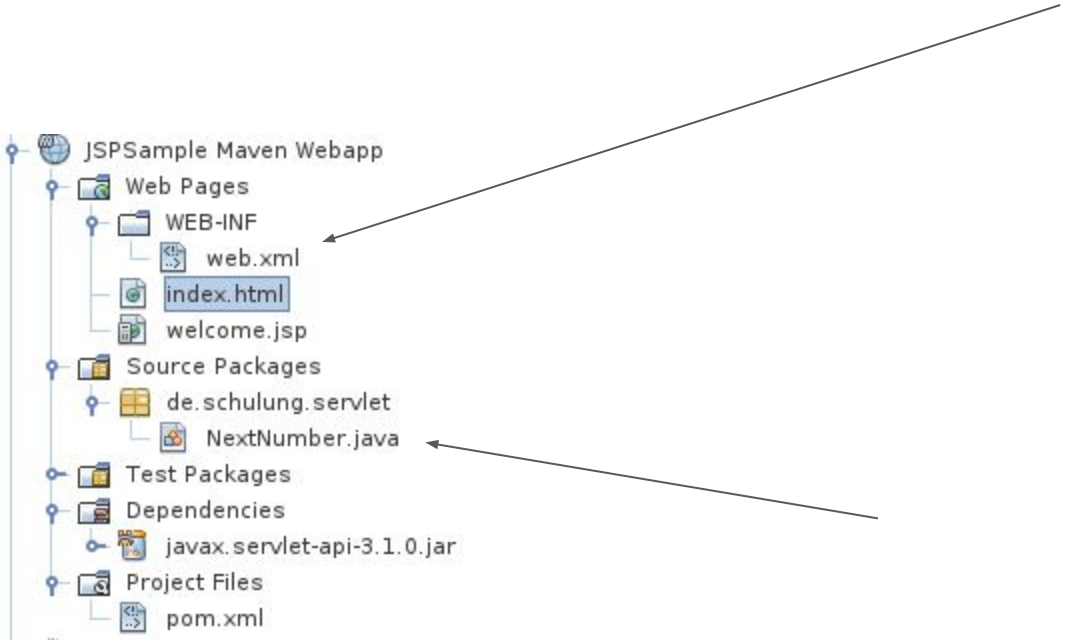
Nächste Nummer Multiplizieren (6)

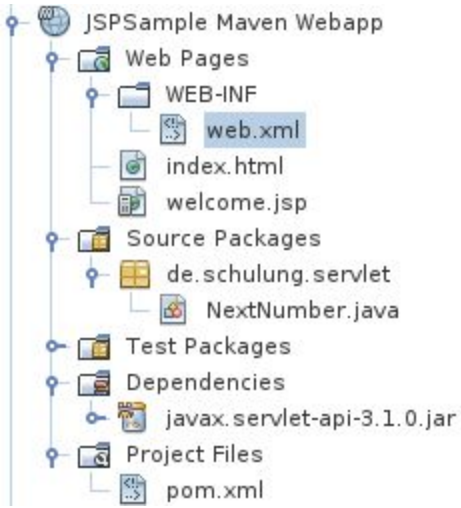


Welcome Max Mustermann

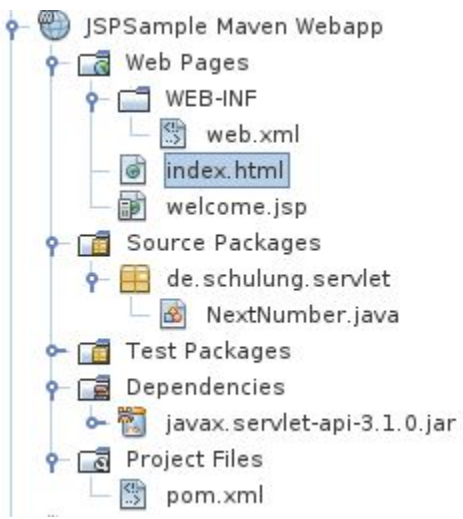
```
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60
```

Nächste Nummer Multiplizieren (6)

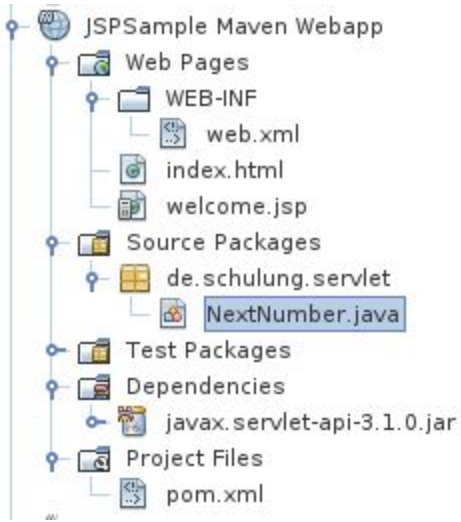

JSPSample Maven Webapp
Web Pages
WEB-INF
web.xml
index.html
welcome.jsp
Source Packages
de.schulung.servlet
NextNumber.java
Test Packages
Dependencies
javax.servlet-api-3.1.0.jar
Project Files
pom.xml




```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>JSP Sample Example</display-name>

    <servlet>
        <description></description>
        <display-name>NextNumber</display-name>
        <servlet-name>NextNumber</servlet-name>
        <servlet-class>de.schulung.servlet.NextNumber</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>NextNumber</servlet-name>
        <url-pattern>/NextNumber</url-pattern>
    </servlet-mapping>
</web-app>
```

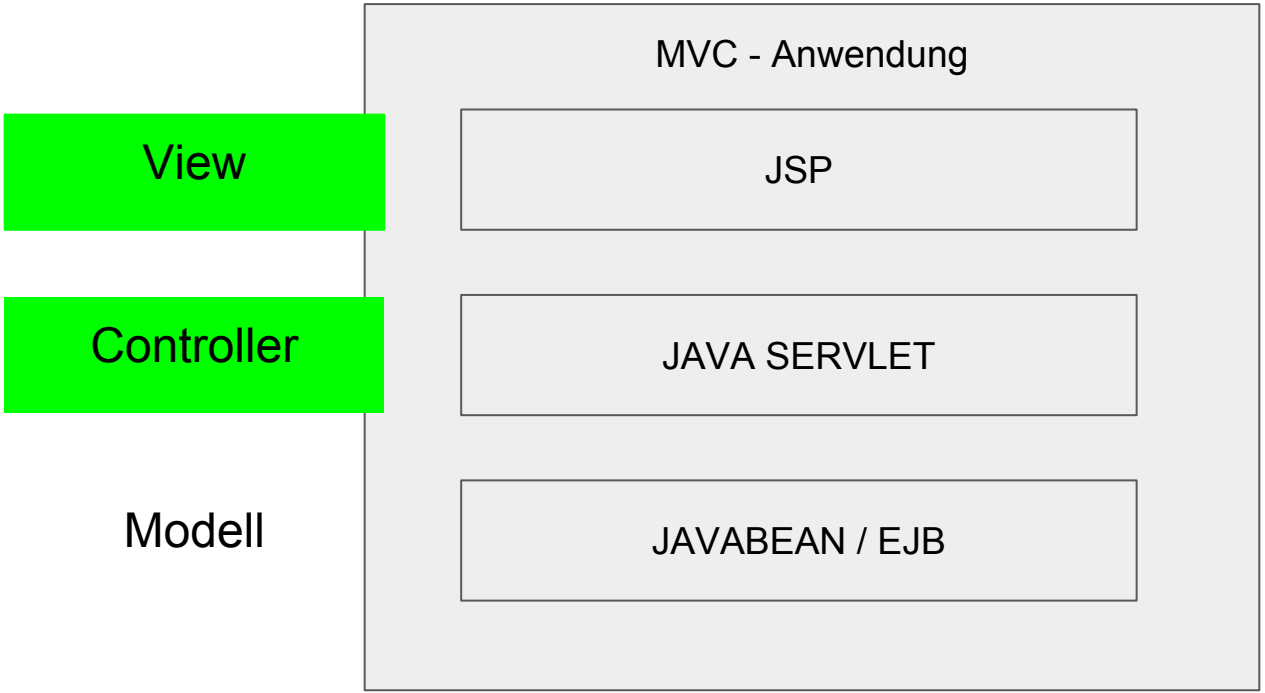



```
<html>
<body>
<h2>Welcome</h2>

<form action="NextNumber" method="get">

Enter username: <input type="text" name="uname" />
<br/>
<br/>
Enter numbr for mulitplication tables: <input type="text" name="number" />
<br/>
<br/>
<input type="submit"/>
<br/>
<br/>

</body>
</html>
```

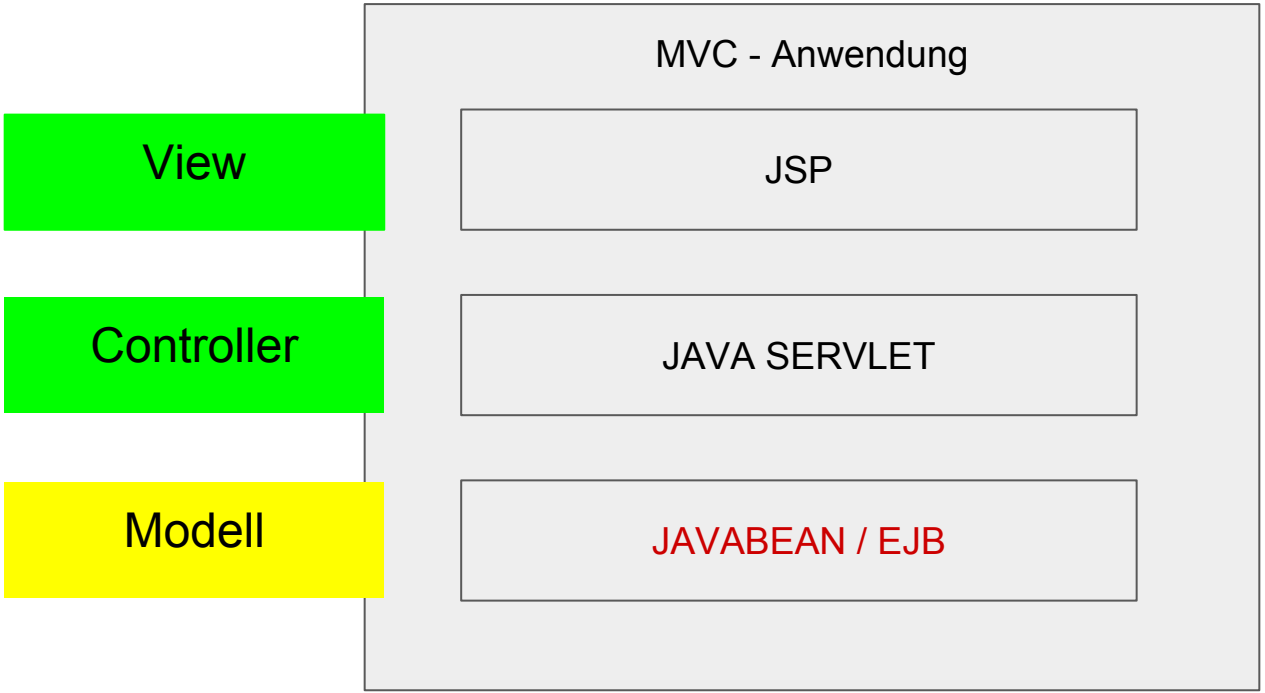



```java
/**
 *
 * @author chdi
 */
public class NextNumber extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {

        String number = request.getParameter("number");

        int nextNumber = Integer.parseInt(number);
        nextNumber++;

        HttpSession session = request.getSession(true);

        session.setAttribute("nextNumber", "" + nextNumber);

        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/welcome.jsp");
        dispatcher.forward(request, response);

    }

}
```

JSP - Servlets im Zusammenspiel



JEE - Das Modell - JAVABEAN & EJB



JAVABEAN / EJB

Enterprise JavaBeans gibt es in mehreren unterschiedlichen Ausprägungen für verschiedene Klassen von Anwendungsfällen.

- Entity Bean - dauerhaften (persistenten) Daten
- Session Bean - Innerhalb einer UserSession
  - unterscheiden in Stateless & Statefull Session Beans
- Message Driven Bean
  - für asynchrone Kommunikation

Entity Bean

Die Entity Bean ist ein Teil der Java Persistence API.

Das bedeutet dass wir unsere Datenbank Struktur in einem POJO Abbilden können.

https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html

```java
@Entity
public class SampleEntity {

    @Id
    private Long id;

    String name;

    public String getName() {
        return name;
    }

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

    int age;

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    Date signupdate;

    public Date getSignupdate() {
        return signupdate;
    }

    public void setSignupdate(Date signupdate) {
        this.signupdate = signupdate;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

import javax.persistence.Entity

@ID -> Primary ID in MYSQL

String Name
int age
Date signupdate

-> Erstellt auf der Datenbank seite Tabellen in dem Format wie im POJO Angegebene

```
24      Account account;
25
26      @OneToOne
27      public Account getAccount () {
28        return account;
29      }
30
31      public void setAccount (Account account) {
32        this.account = account;
33      }
34
35      Employee salesRep;
36
37      @ManyToOne
38      public Employee getSalesRep () {
39        return salesRep;
40      }
41
42      public void setSalesRep (Employee salesRep) {
43        this.salesRep = salesRep;
44      }
45
46      Vector <Order> orders;
47
48      @OneToMany
49      public Vector <Order> getOrders () {
50        return orders;
51      }
52
53      public void setOrders (Vector <Order> orders) {
54        this.orders = orders;
55      }
56
57  }
```

Andere JAVA Objekte

1 : 1 Verknüpfung

n : 1 Verknüpfung

1 : n Verknüpfung

Einfaches Arbeiten auf der Datenbank !

```java
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**
 *
 * @author chdi
 */
public class EM {

    @PersistenceContext
    EntityManager em;

    Customer customer = new Customer();

    public void datenbank_Access() {

        // Speichert neuen Customer
        em.persist(customer);

        // Erhöt das Alter um 1 Jahr und Speichert dies gleichzeitig in der DB
        customer.setAge(customer.getAge() + 1);

        //Löscht ein Customer aus der DB
        em.remove(customer);

        // Holt alle customer einträge die jünger sind als 30 aus der DB
        List<Customer> customers = (List<Customer>) em.createQuery("select c from Customer where c.age > 30");
    }
}
```

Session Bean

- Session Beans bildet Vorgänge ab die der User im System durchführt
- Man unterscheidet die Session Beans in
  - zustandslose (statless)
  - zustandsbehaftete (stateful)

Interface :

https://docs.oracle.com/javaee/7/api/javax/ejb/SessionBean.html

Stateful Session Bean

- Eine zustandsbehaftete Session Bean hat ein eigenes Gedächtnis.
- Sie kann Informationen aus einem Methodenaufruf speichern, damit sie bei einem späteren Aufruf einer anderen (oder der gleichen) Methode wieder zur Verfügung stehen.
- Die Zustandsbehaftung wird durch die Vergabe einer eindeutigen ID umgesetzt, über diese ID können die zustandsbehafteten (stateful) Session Beans unterschieden werden.

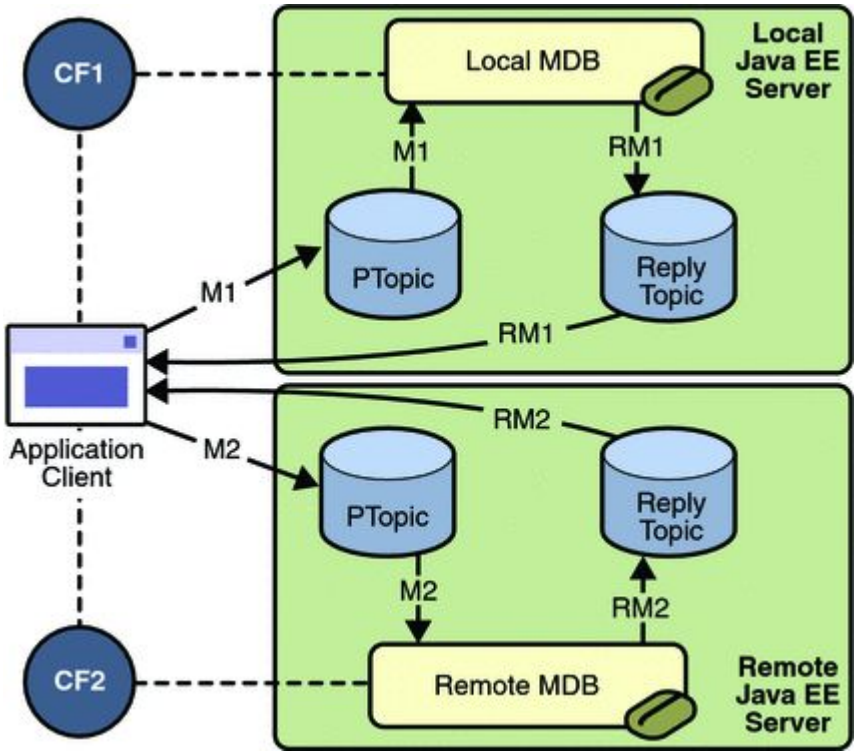
Stateless Session Bean

- zustandslosen Session Bean benötigt bei jedem Aufruf alle Informationen als Parameter wieder übergeben.
- Da die zustandslose Session Bean keine Informationen speichern kann. Ist sie von anderen Session Beans der gleichen Klasse unterscheidbar.
- Somit ist der Aufruf nicht asynchron zuordnungsbar.
- Zustandslose Beans sollte somit immer im synchronen Fluss oder bei Fire and Forget genutzt werden

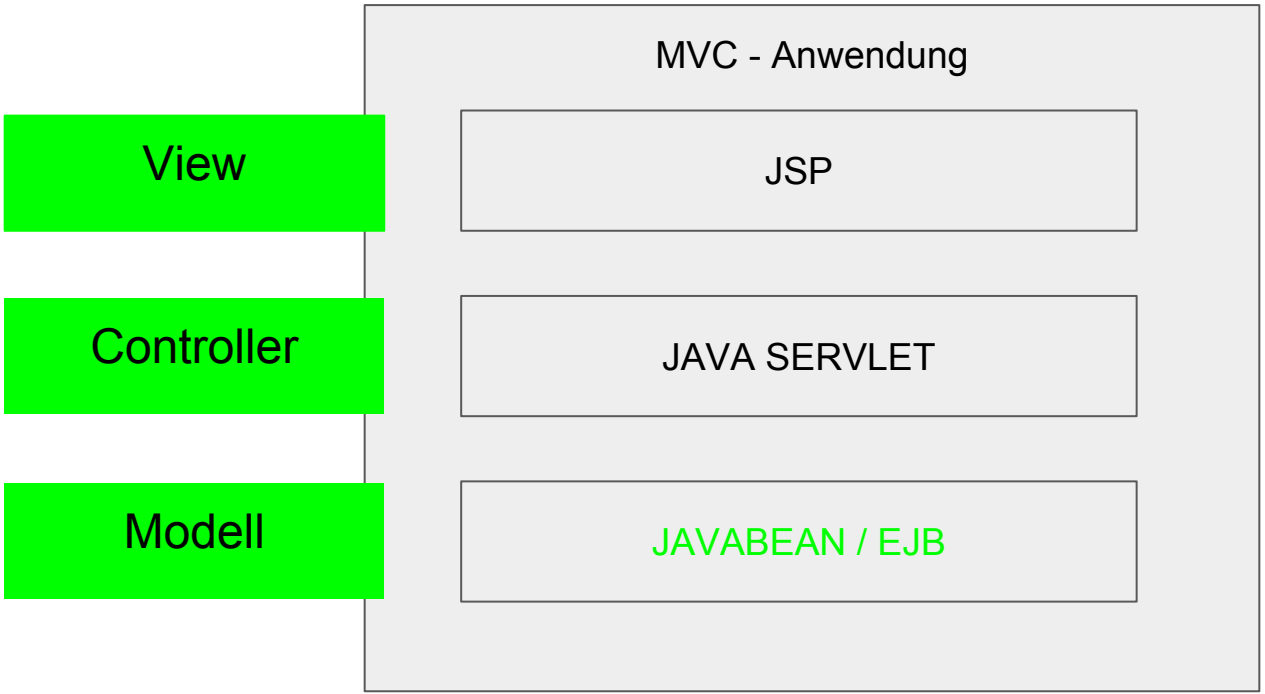
Message Driven Bean

- Message Driven Beans sind diejenigen Komponenten, die EJB-Systeme für asynchrone Kommunikation zugänglich machen.
- häufig für die Kommunikation mit Legacy-Systemen genutzt
- Auch für die Ausführung von klassischerweise asynchron auszuführenden Operationen (z. B. dem Verschicken einer Mail)

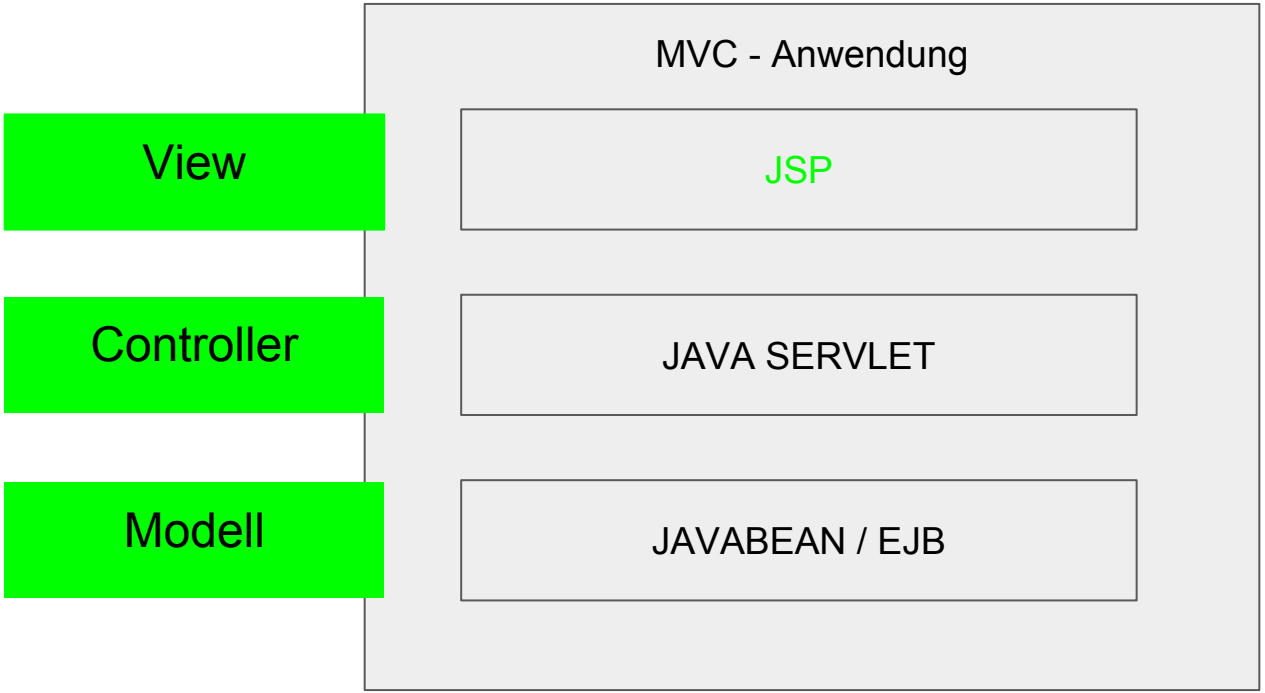
Wie sieht das ganze für uns aus ?



Quelle : docs.oracle.com



Tieferes Know How im Bereich EJB dann in den weiteren JEE Schulungen


MVC - Anwendung
View
JSP
Controller
JAVA SERVLET
Modell
JAVABEAN / EJB

Was ist hier Problematisch / unschön ?

```
<%@ 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>Multiplication tables</title>
</head>
<body>

<h3>Welcome <%=request.getParameter("uname") %></h3>
<pre>
<%
String value = request.getParameter("number");
try
{
        Integer number = Integer.parseInt(value);
        for(int i=1;i<=10;i++)
        {
                out.println(number + " * " + i +" = " + (number*i)  );
        }
}
catch(NumberFormatException e)
{
        out.println("Invalid number");
}
%>
<pre>
<br/>
<br/>


</body>
</html>
```

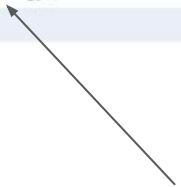
JSTL - die Lösung

- JavaServer Pages Standard Tag Library
- Sammlung von vielen Custom-Tag-Bibliotheken
- JSTL wird interpretiert und nicht kompiliert.
- Ermöglicht Änderungen der HTML Dateien zur Laufzeit
- JSTL ist nicht wie Struts an Architektur-Paradigmen wie MVC gebunden

Was Bedeutet das in der Praxis

Wir machen unserer HTML Seite JSTL bekannt

```
<%--
    Document   : welcomejstl
    Created on : 30.11.2017, 10:13:58
    Author     : chdi
--%>

<%@ 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>Multiplication tables</title>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
</head>
```



Bauen sie die Application selbständig um

Schleife sieht nun wie Folgt aus

```
<c:catch var ="catchtheException">
   <% int x = 2/0;%>
</c:catch>
```

Eine Schleife

```
<c:forEach var="j" begin="1" end="3">
   Item <c:out value="${j}"/><p>
</c:forEach>
```

```
<%--
    Document   : welcomejstl
    Created on : 30.11.2017, 10:13:58
    Author     : chdi
--%>

<%@ 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>Multiplication tables</title>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
</head>
<body>
<h3>Welcome <c:out value="${param.number}">></h3>
<pre>
<p>
    <c:catch var="Invalid number">
        <c:forEach var="j" begin="1" end="10">
           <c:out value="${param.number} * ${j} = ${param.number * j} "/>
        </c:forEach>
    </c:catch>
</p>
<pre>

<a href="http://localhost:8080/JSPSample/welcomejstl.jsp?uname=${param.uname}&number=${sessionScope.nextNumber}">
Nächste Nummer Multiplizieren <c:out value="${sessionScope.nextNumber}"></c:out>
</a>
<br/>
<br/>


</body>
</html>
```

JSF

Java Server Faces (JSF) ist ein Java-basiertes Webanwendungs-Framework, das die Entwicklung Integration von webbasierten Benutzeroberflächen vereinfachen soll.

JavaServer Faces ist eine standardisierte Frontend Technologie , die in einer Spezifikation durch den Java Community Process formalisiert wurde.

Vorteile

JSF reduziert den Aufwand für das Erstellen und Verwalten von Anwendungen, die auf einem Java-Anwendungsserver ausgeführt werden und die Anwendung Benutzeroberfläche auf einem Ziel Client bereitstellen. JSF erleichtert die Entwicklung von Webanwendungen durch

- Bereitstellung wiederverwendbarer UI-Komponenten
- Einfache Datenübertragung zwischen UI-Komponenten

- Verwalten des Benutzeroberflächen Status über mehrere Server Anforderungen

- Implementierung von benutzerdefinierten Komponenten aktivieren
- Clientseitiges Ereignis mit serverseitigem Anwendungscode verbinden

Example

JSF example

Name Eingeben Rafael

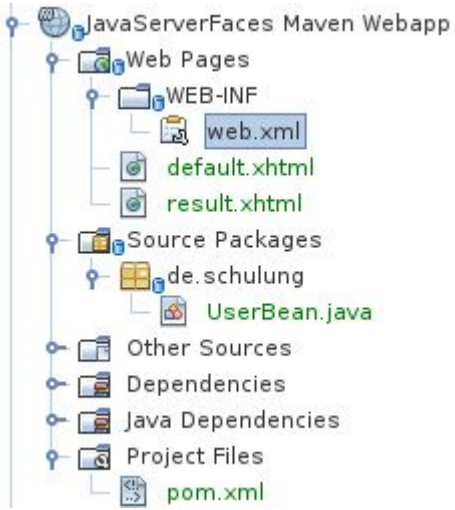
Absenden

localhost:8080/JavaServerFaces/faces/default.xhtml

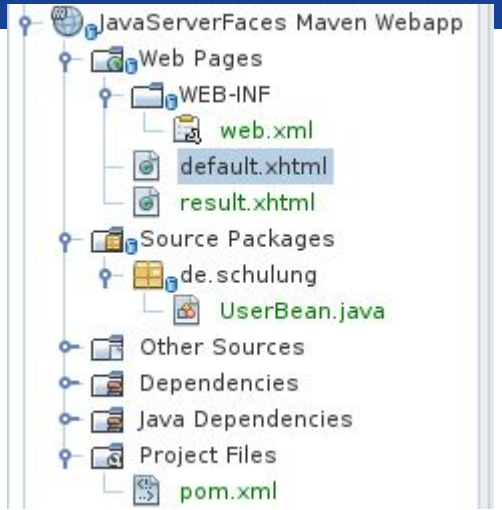
Na nu wer bischn du ?

Ich bin, Rafael. und komme aus Karlsruhe.

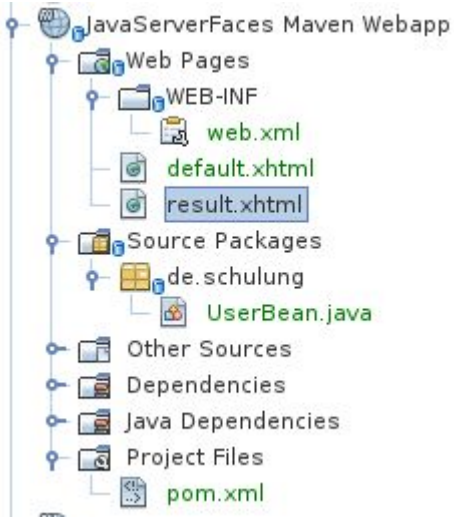
Projekt Struktur



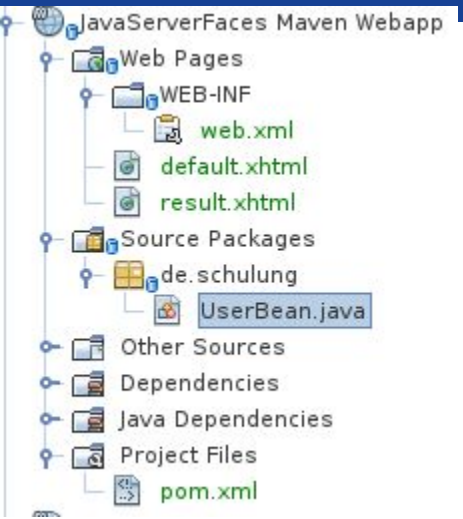

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         id="WebApp_ID" version="2.5">

  <display-name>JavaServerFaces</display-name>

  <!-- Change to "Production" when you are ready to deploy -->
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>

  <!-- Welcome page -->
  <welcome-file-list>
    <welcome-file>faces/default.xhtml</welcome-file>
  </welcome-file-list>

  <!-- JSF mapping -->
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <!-- Map these files with JSF -->
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
  </servlet-mapping>

</web-app>
```




```xml
<?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:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      >

    <h:body>

        <h1>JSF example</h1>

            <h:form id="form">

                Name Eingeben
                <h:inputText size="10" value="#{user.name}" />

                <br /><br />

                <h:commandButton id="submitButton"
                        value="Absenden" action="#{user.outcome}">

                    <f:param name="country" value="Karlsruhe" />

                </h:commandButton>

            </h:form>

    </h:body>
</html>
```




```
<?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:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      >

    <h:body>

        <h1>JSF 2 param example</h1>

                <h3>
                <h:outputFormat value="Hallo ,{0}. Du kommst aus {1}.">
                        <f:param value="#{user.name}" />
                        <f:param value="#{user.country}" />
                </h:outputFormat>
                </h3>

    </h:body>

</html>
```




```java
package de.schulung;

import java.util.Map;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;

@ManagedBean(name="user")
@SessionScoped
public class UserBean{

	public String name;
	public String country;

	public String outcome(){

		FacesContext fc = FacesContext.getCurrentInstance();
		this.country = getCountryParam(fc);

		return "result";
	}

	//get value from "f:param"
	public String getCountryParam(FacesContext fc){

		Map<String,String> params = fc.getExternalContext().getRequestParameterMap();
		return params.get("country");

	}

	public String getName() {
		return name;
	}

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

	public String getCountry() {
		return country;
	}

	public void setCountry(String country) {
		this.country = country;
	}

}
```

JSF example

Name Eingeben Rafael

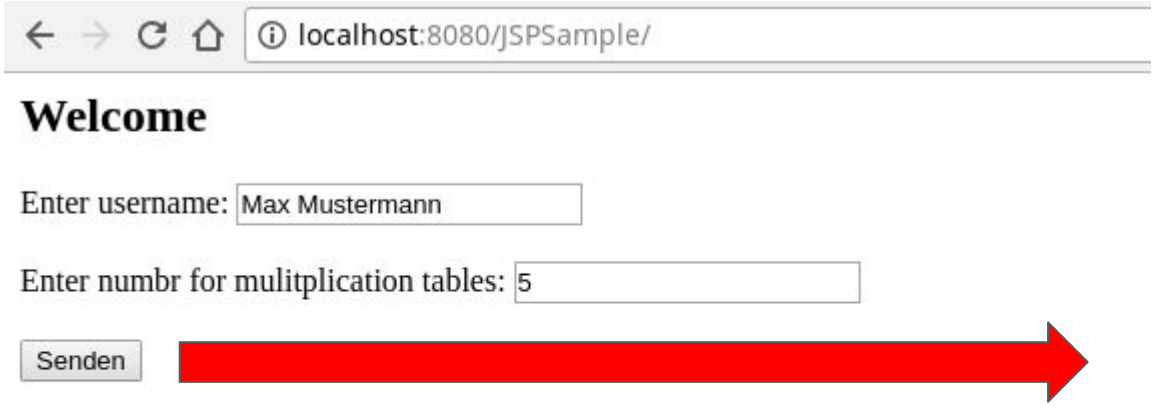
Absenden

localhost:8080/JavaServerFaces/faces/default.xh

Na nu wer bischn du ?

Ich bin, Rafael. und komme aus Karlsruhe.

Aufgabe : Bauen sie folgende Application nach



Welcome Max Mustermann

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```