

Webanwendungen Grundlagen

Einführung in die Web-Programmierung mit Java

Prof. DI Dr. Erich Gams
htl-wels.e.gams@eduhi.at

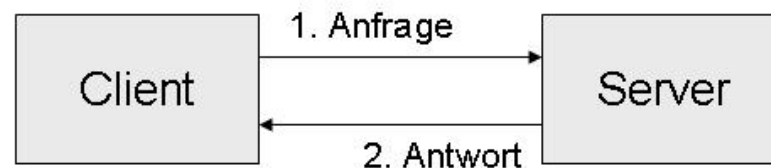


Übersicht ➡ Was lernen wir?

- Schichtenarchitektur
- Was sind Servlets?
- Aufgaben eines Servlets
- Servlet-Container
- Grundstruktur von Servlets
- Lebenszyklus eines Servlets

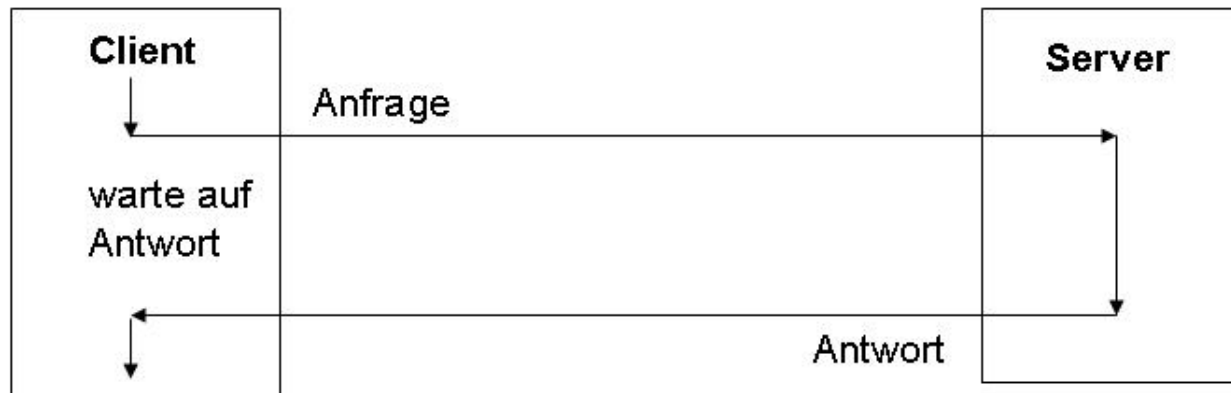
Client/Server Modell

- Client und Server bezeichnet die Beziehung in der 2 Programme zueinander stehen. Client stellt Anfrage und der Server behandelt die Anfrage.

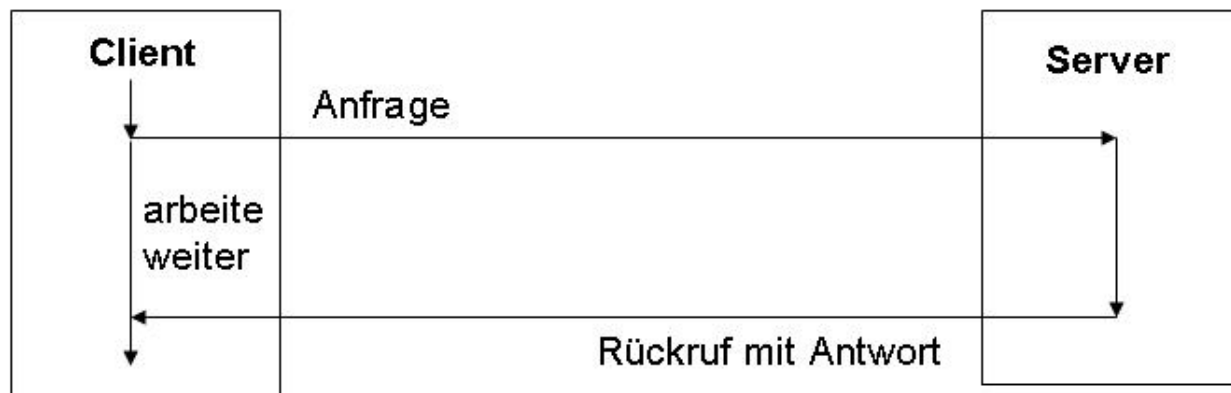


Interaktion zwischen Client und Server

synchrone Kommunikation



asynchrone Kommunikation



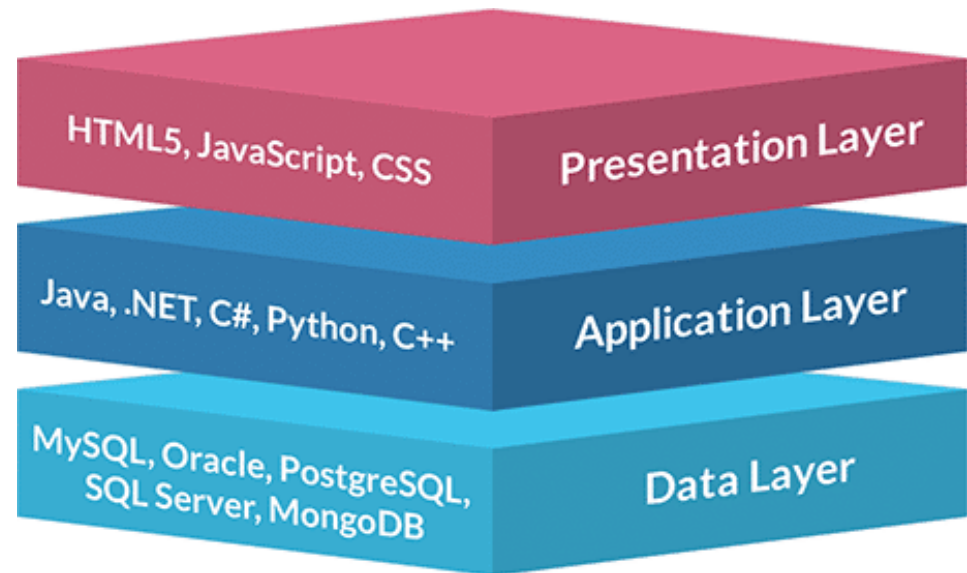
Mehrstufige Architekturen (n-tier applications)

- Jede Anwendung kann in 3 wesentliche Schichten unterteilt werden.

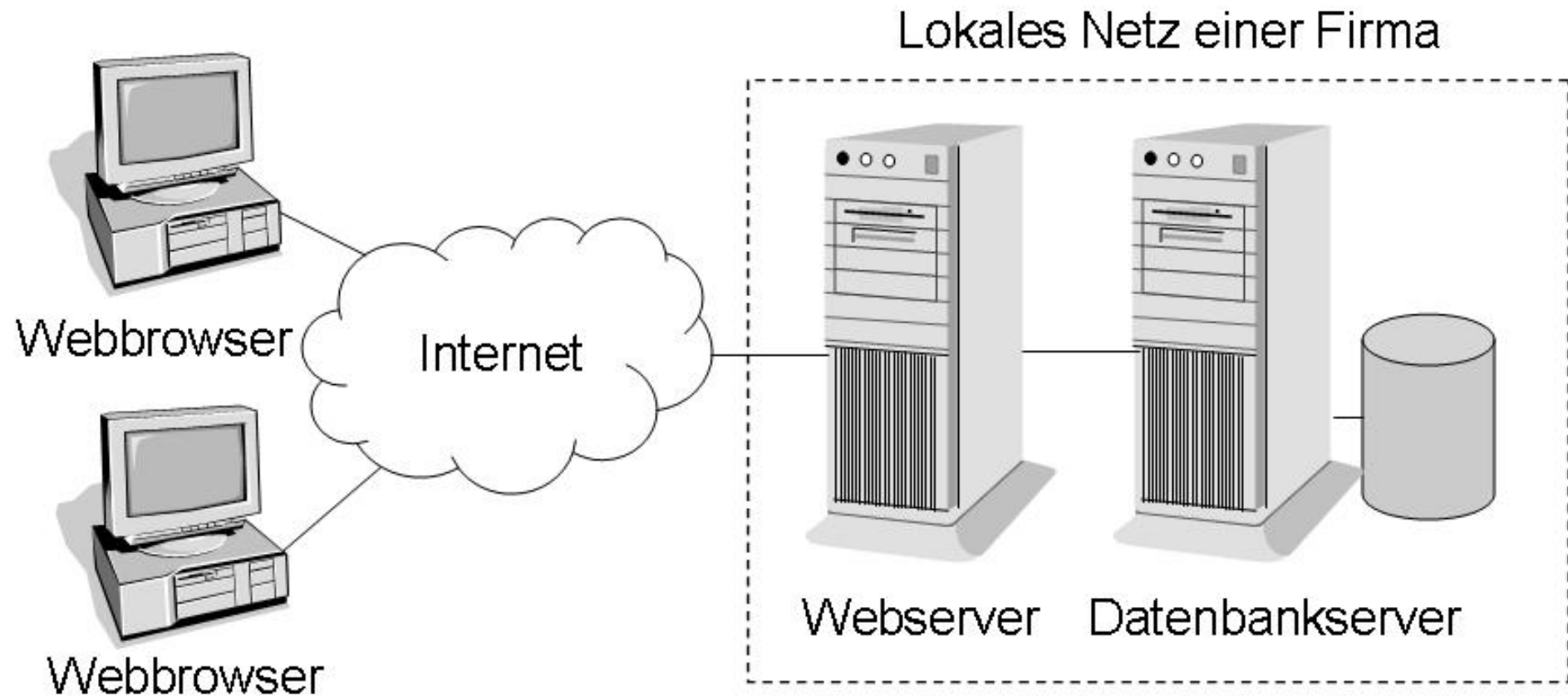
1. Präsentation
2. Verarbeitung
3. Datenhaltung

Schichtenarchitektur

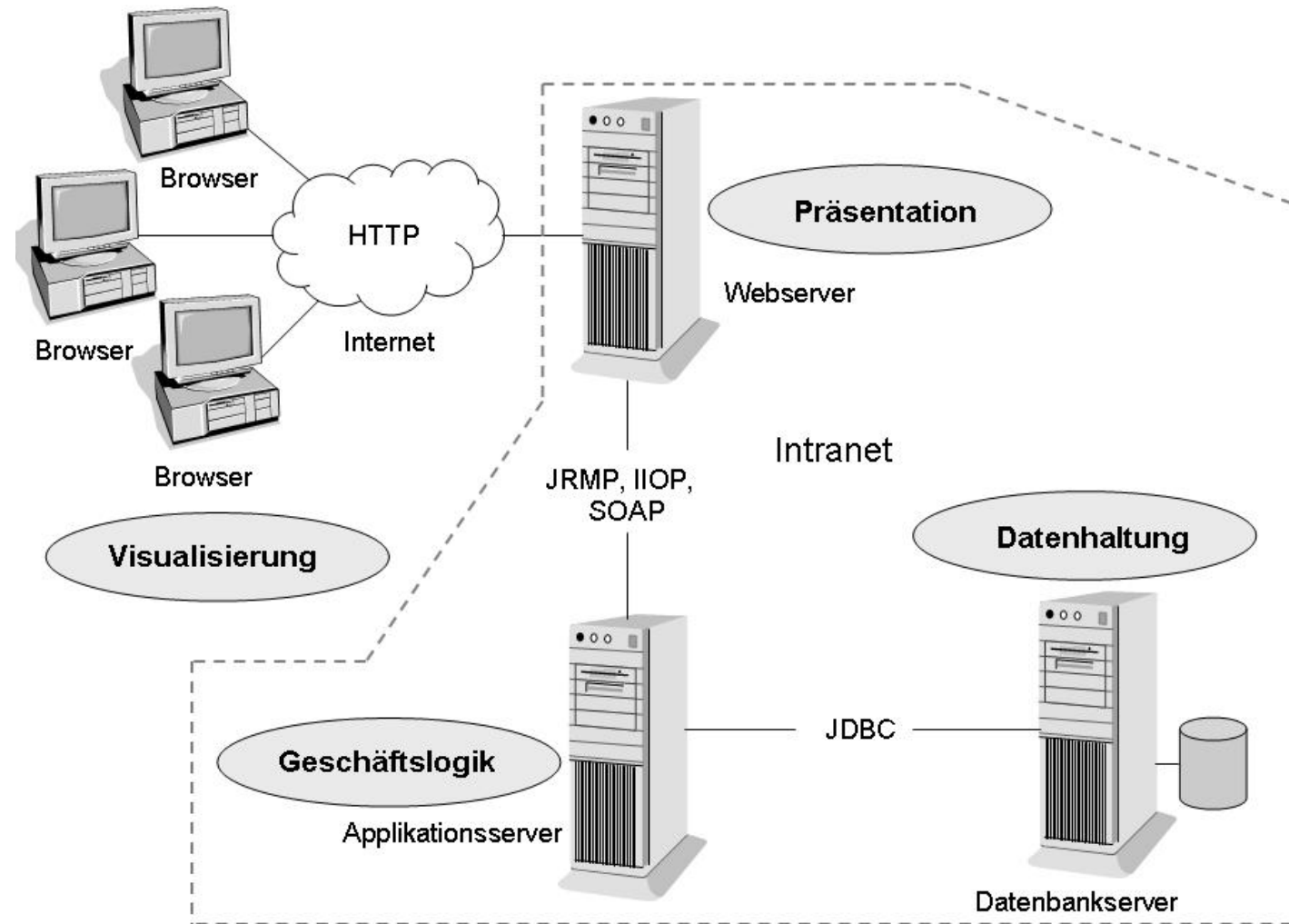
- Präsentationsschicht (presentation layer)
 - Steuerung des Dialogs mit dem Benutzer, Daten- und Befehlsübermittlung
- Verarbeitung (business / application layer)
 - Verarbeitungslogik, Erzielung des eigentlichen Nutzens der Anwendung
- Datenhaltung (persistence / data layer)
 - Ablage und Zugriff auf Daten
- Beispiel: Webbrowser, Webserver, Datenbankserver



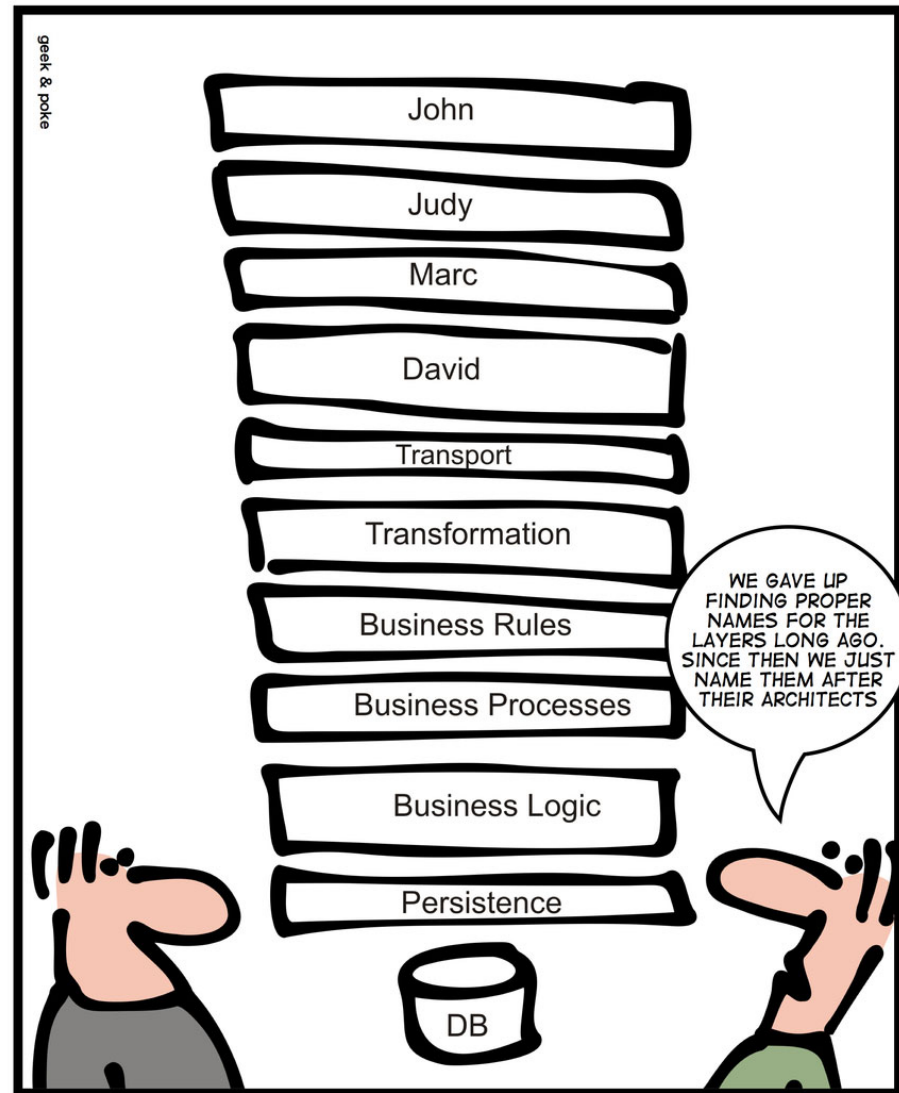
3-Tier Application



N-Tier Application



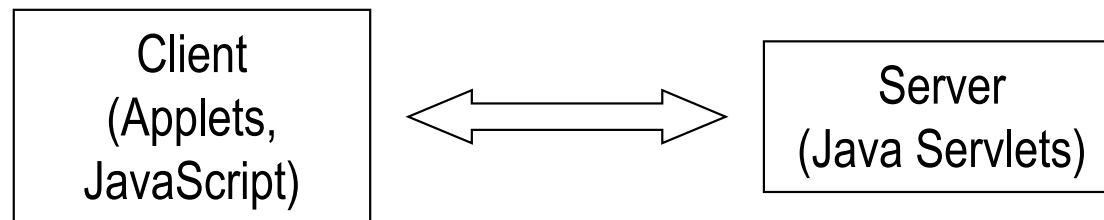
n-tier application



A GOOD ARCHITECT LEAVES A FOOTPRINT

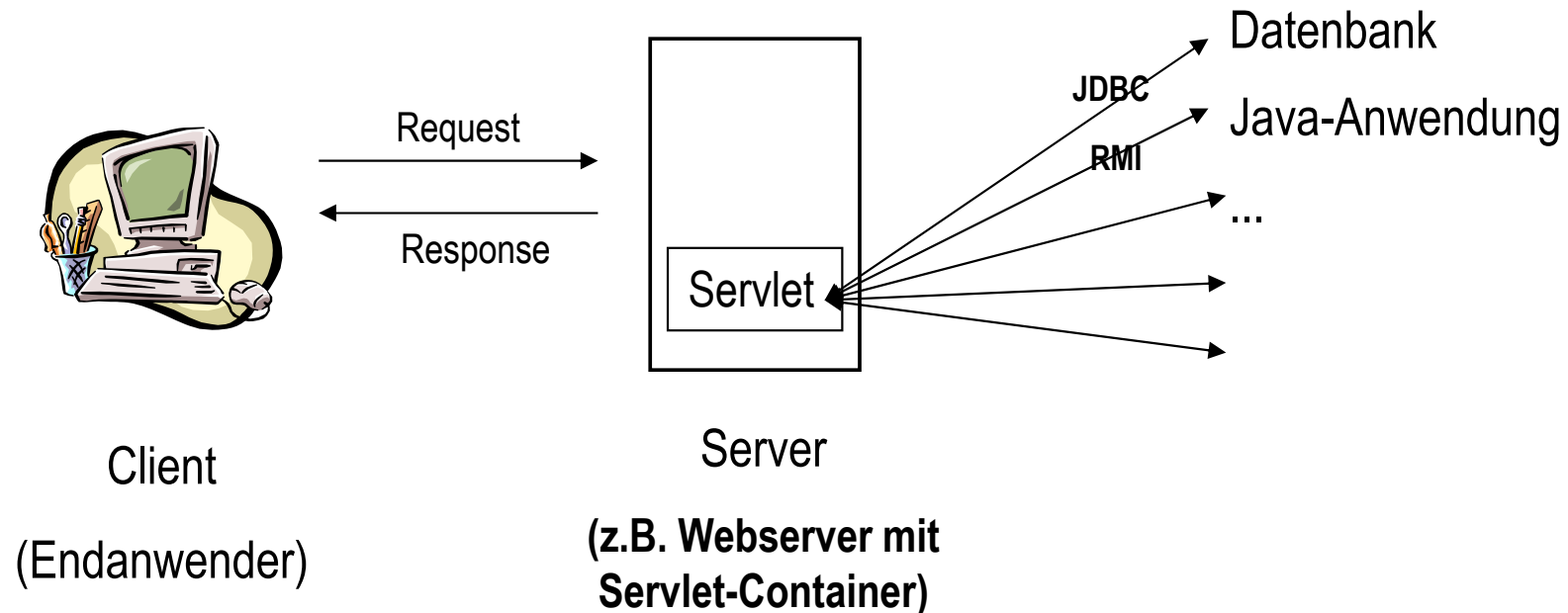
Was sind Servlets?

- Java-Programme, die auf einem Web- oder Anwendungsserver ausgeführt werden.



- dienen der dynamischen Generierung von Webseiten / Webcontent (Basis für alle Java Server Frameworks)
- plattformunabhängig durch Verwendung der Java-Technologie
- prinzipiell protokollunabhängig, werden allerdings meistens im Zusammenhang mit HTTP verwendet

Aufgaben eines Servlets



1. Vom Client gesendete, explizite Daten lesen
2. Vom Browser implizit mit der HTTP-Anfrage gesendete Daten lesen
3. Ergebnisse generieren (mit Hilfe der in Java zur Verfügung stehenden Werkzeuge)
4. Konkrete Daten an den Client zurücksenden
5. Implizite Antwortdaten an den Client senden

Grundstruktur von Servlets (1)

- Servlets werden normalerweise von **HttpServlet** abgeleitet
- Überschreiben die Methoden **doGet()** und **doPost()**
 - **doGet()** und **doPost()** nehmen jeweils 2 Parameter entgegen:
 - **HttpServletRequest**
 - ermöglicht Zugriff auf alle eingehenden Daten
 - **HttpServletResponse**
 - Ermöglicht die Spezifikation von ausgehenden Informationen
 - Beinhaltet den **PrintWriter**, mit dem Dokumentinhalt an den Client zurückgesendet werden kann
 - Lösen 2 Ausnahmen aus:
 - **ServletException**
 - **IOException**

Grundstruktur von Servlets (3)

- **PrintWriter** erfordert den import von `java.io`
- **HttpServlet** erfordert den import von `javax.servlet`
- **HttpServletRequest / HttpServletResponse** erfordern den import von `javax.servlet.http`

Plain Text Servlet Beispiel

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

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}
```

Annotationen (oder web.xml)

```
package servlets;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/HelloWorldServlet")
public class HelloWorldServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello HTL-Wels!");
    }
}
```

- **@WebServlet("/address")**
 - Die URL relativ zum Applikationsnamen.
- **doGet**
 - Aufruf bei HTTP GET request.
- **HttpServletRequest**
 - Beinhaltet alle Informationen, die vom Browser kommen.
- **HttpServletResponse**
 - Beinhaltet alles was zum Browser gesendet wird. Meist mit PrintWriter realisiert.

Ein HTML-generierendes Servlet (1)

- **Mitteilung an den Browser**
 - `response.setContentType("text/html");`
- **Verwende println Statements, um eine HTML Seite zu erstellen**
 - Print Statements müssen HTML tags enthalten
- **Überprüfe deine HTML mit einem Syntax validator**
 - <http://validator.w3.org/>
 - <http://www.htmlhelp.com/tools/validator/>

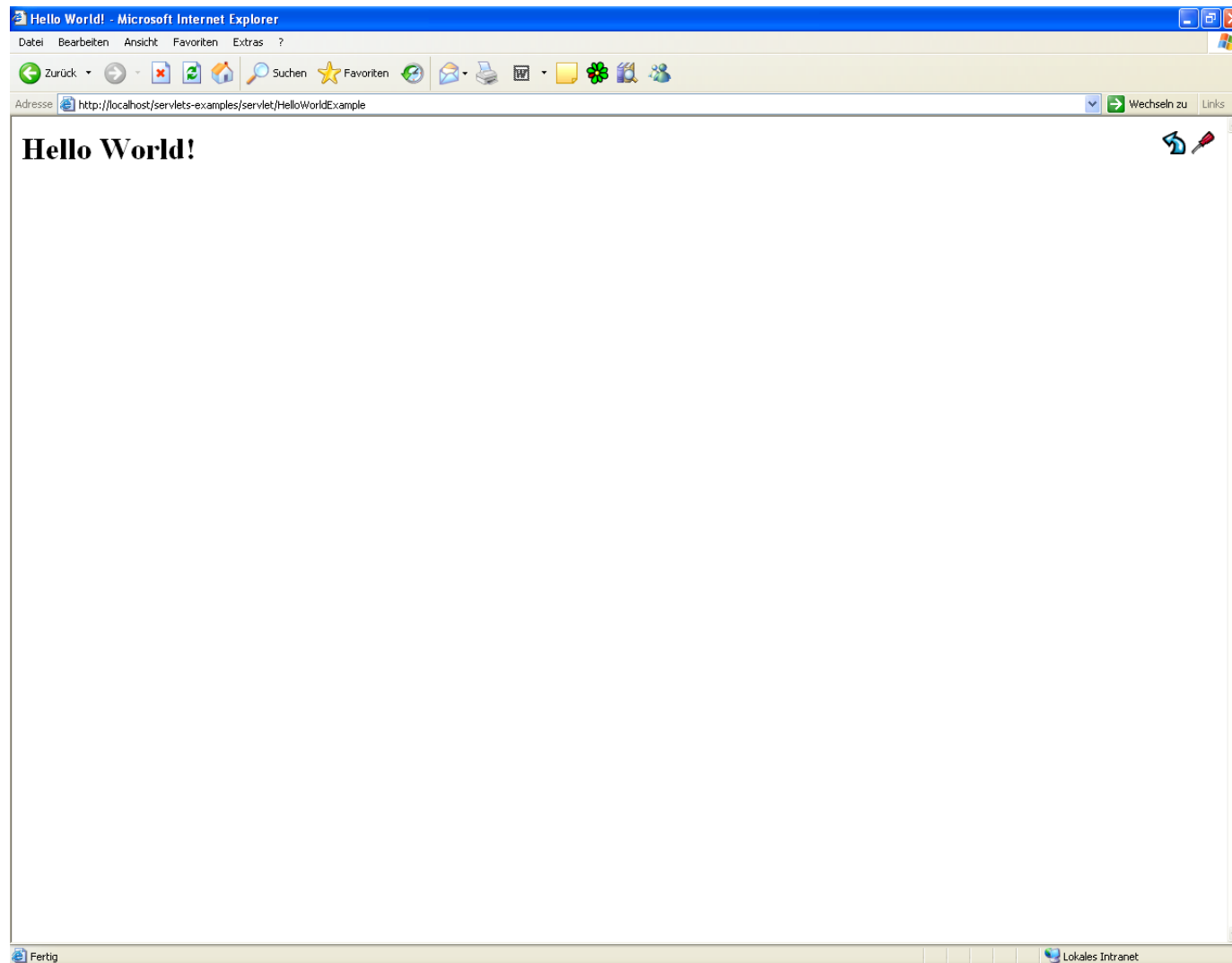
Ein HTML-generierendes Servlet (2)

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

/** Simple servlet used to test server. */

public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
            \"Transitional//EN\">\n";
        out.println(docType +
                    "<HTML>\n" +
                    "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
                    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                    "<H1>Hello</H1>\n" +
                    "</BODY></HTML>");
    }
}
```

Ergebnis: Hello World



Servlets Utilities

```
package coreservlets;

import javax.servlet.*;
import javax.servlet.http.*;

/** Some simple time savers. Note that most are static methods. */

public class ServletUtilities {
    public static final String DOCTYPE =
        "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " +
        "Transitional//EN">";

    public static String headWithTitle(String title) {
        return(DOCTYPE + "\n" +
            "<HTML>\n" +
            "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
    }

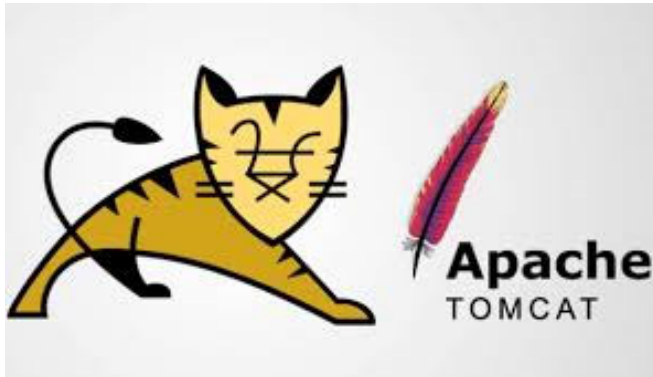
    ...
}
```

```
@WebServlet("/test-with-utils")
public class TestServlet2 extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Test Servlet with Utilities";
        out.println(ServletUtilities.headWithTitle(title) + "<body
bgcolor=\"#fdf5e6\">\n" + "<h1>" + title + "</h1>\n" + "<p>Simple servlet for
testing.</p>\n" + "</body></html>");
    }
}
```

Servlet-Container

- Voraussetzung für die Nutzung von Web-Applikationen:
Vorhandensein eines Servlet-Containers
(auch als Servlet-Engine oder Web-Container bezeichnet) auf dem Server
 - Implementiert die API-Spezifikationen Servlet Servlet 4.0
 - Zu Entwicklungszwecken in der Regel als eigenständiger Desktop-Entwicklungsserver eingerichtet
 - e.g. Tomcat oder
 - Applikationsserver: TomEE, Apache Geronimo, Glassfish, JBoss EAP, Oracle WebLogic, usw.

Servlet-Container



GlassFish

Deployment Descriptor: web.xml

- Die Struktur jeder Webapplikation wird mit Hilfe der XML-Datei web.xml beschrieben.
- Zwei wichtige Elemente beim Arbeiten mit Servlets sind
 - `servlet`
 - `servlet-mapping`

Deployment Descriptor: web.xml

- **Java code**

package myPackage; ...

public class MyServlet extends HttpServlet { ... }

- **web.xml entry (in <web-app...>...</web-app>)**

- Vergeben eines Namens an ein Servlet

<servlet>

<servlet-name>MyName</servlet-name>

<servlet-class>myPackage.MyServlet</servlet-class>

</servlet>

- Adress an ein Servlet übergeben (URL mapping)

<servlet-mapping>

<servlet-name>MyName</servlet-name>

<url-pattern>/MyAddress</url-pattern>

</servlet-mapping>

- **URL:** http://hostname/webappPrefix/MyAddress

Web-Apps ohne Verwendung der web.xml

- **Die Adresse mit @WebServlet festlegen**

`@WebServlet("/my-address")`

`@WebServlet(name="HelloHTLServlet", urlPatterns =
"/HelloHTLServlet")`

`public class MyServlet extends HttpServlet { ... }`

In Run Configurations einzustellen

- URL:

<http://hostName/appContext/my-address>



- Können in web.xml überschrieben werden!

```
<welcome-file-list>  
  <welcome-file>index.html</welcome-file>  
  <welcome-file>index.htm</welcome-file>  
  <welcome-file>index.jsp</welcome-file>  
</welcome-file-list>
```

Startpunkt festlegen

The image shows two screenshots from the Eclipse IDE. The left screenshot displays the 'Server' tab of the 'Run/Debug Configurations' dialog, where the 'Application context' is set to '/MyTest'. The right screenshot shows the 'Deployment' tab of the same dialog, where the 'URL' is set to 'http://localhost:8080/MyTest/HelloWorldServlet'. Both fields are circled in red.

Server Deployment Logs Code Co

Deploy at the server startup

Servlet-HelloWorld:war exploded

Application context: /MyTest

Run/Debug Configurations

Name: TomEE 9.0.22

Server Deployment Logs Code Coverage Sta

Application server: TomEE 9.0.22

Open browser

☒ After launch Default ... ☐ wi

URL: http://localhost:8080/MyTest/HelloWorldServlet

VM options:

On 'Update' action: Update classes and resources

On frame deactivation: Do nothing

JRE: Default (1.8 - project SDK)

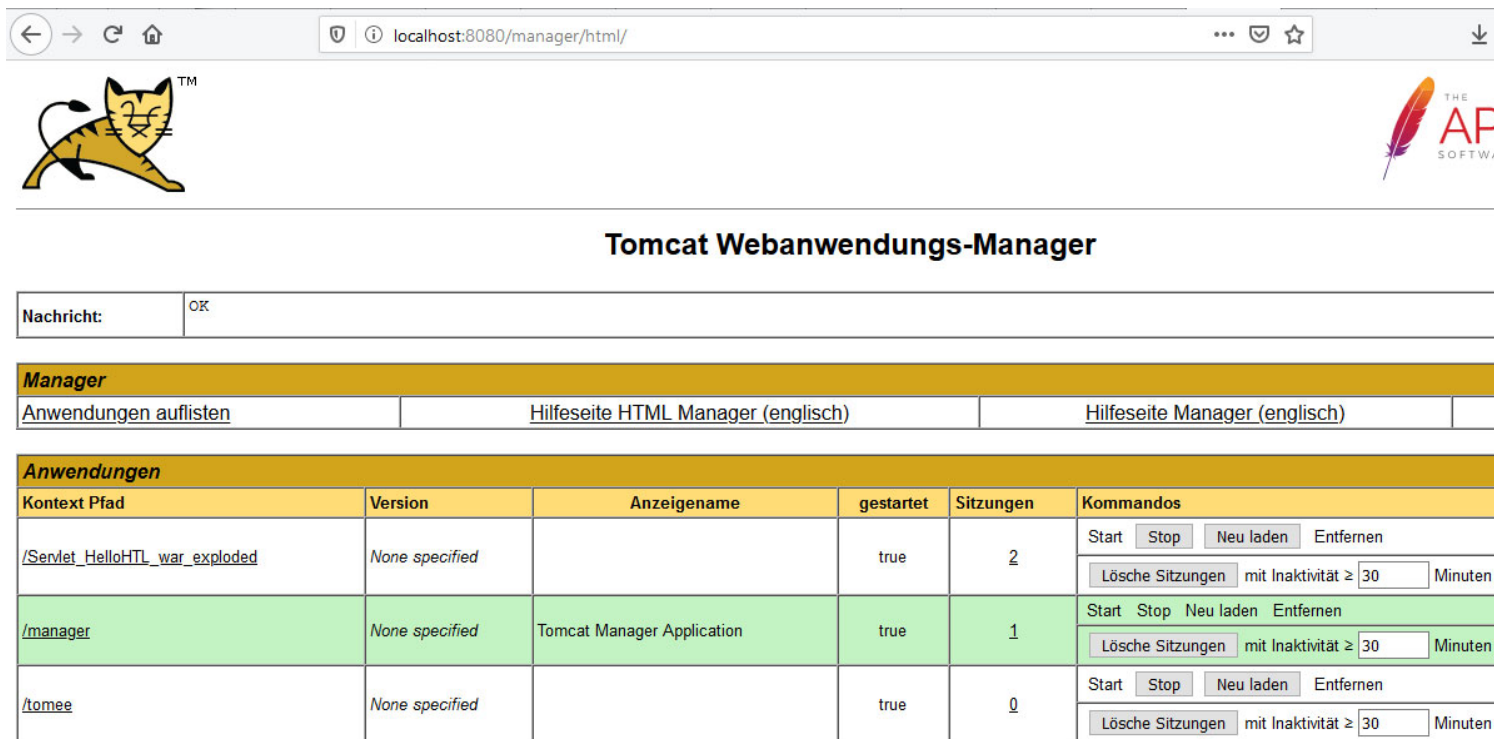
Startpunkt festlegen (Workarounds)

- Run configurations: Servlet(=URL Pattern) eintragen
- `@WebServlet(name="HelloHTLServlet", urlPatterns = "/*")`
- `@WebServlet(name="HelloHTLServlet", urlPatterns = "/index.html")`

- ../config/server.xml
- Connector port
 - Port an dem die Requests geschickt werden und von dem die responses ausgehen
 - Standard bei Tomcat **8080**
 - Aufruf: localhost:8080

Management Konsole /manager

- Benutzer einstellen: Tomcat-users.xml



Tomcat Webanwendungs-Manager

Nachricht: OK

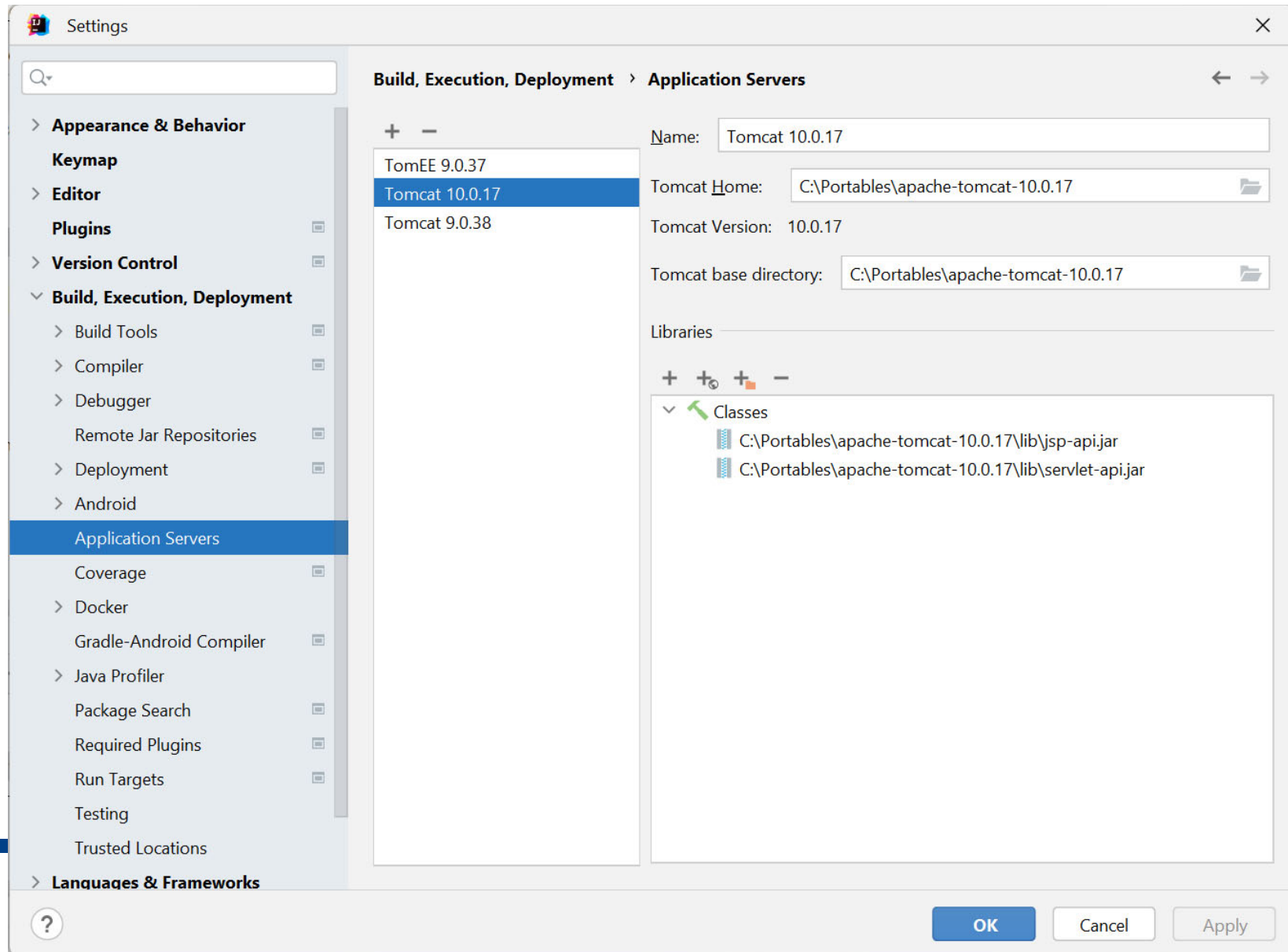
Manager

[Anwendungen auflisten](#) | [Hilfeseite HTML Manager \(englisch\)](#) | [Hilfeseite Manager \(englisch\)](#)

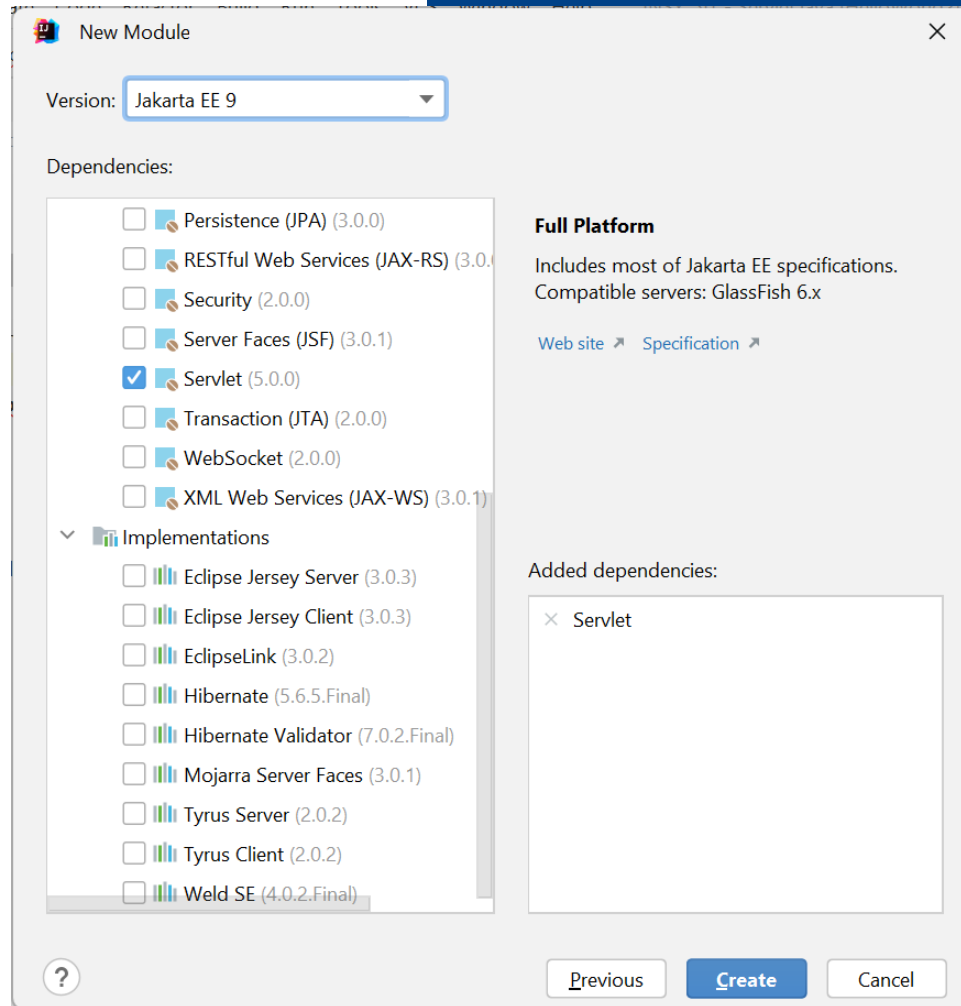
Anwendungen

| Kontext Pfad | Version | Anzeigename | gestartet | Sitzungen | Kommandos |
|--|----------------|----------------------------|-----------|-----------|---|
| /Servlet_HelloHTL_war_exploded | None specified | | true | 2 | Start Stop Neu laden Entfernen Lösche Sitzungen mit Inaktivität ≥ 30 Minuten |
| /manager | None specified | Tomcat Manager Application | true | 1 | Start Stop Neu laden Entfernen Lösche Sitzungen mit Inaktivität ≥ 30 Minuten |
| /tomee | None specified | | true | 0 | Start Stop Neu laden Entfernen Lösche Sitzungen mit Inaktivität ≥ 30 Minuten |

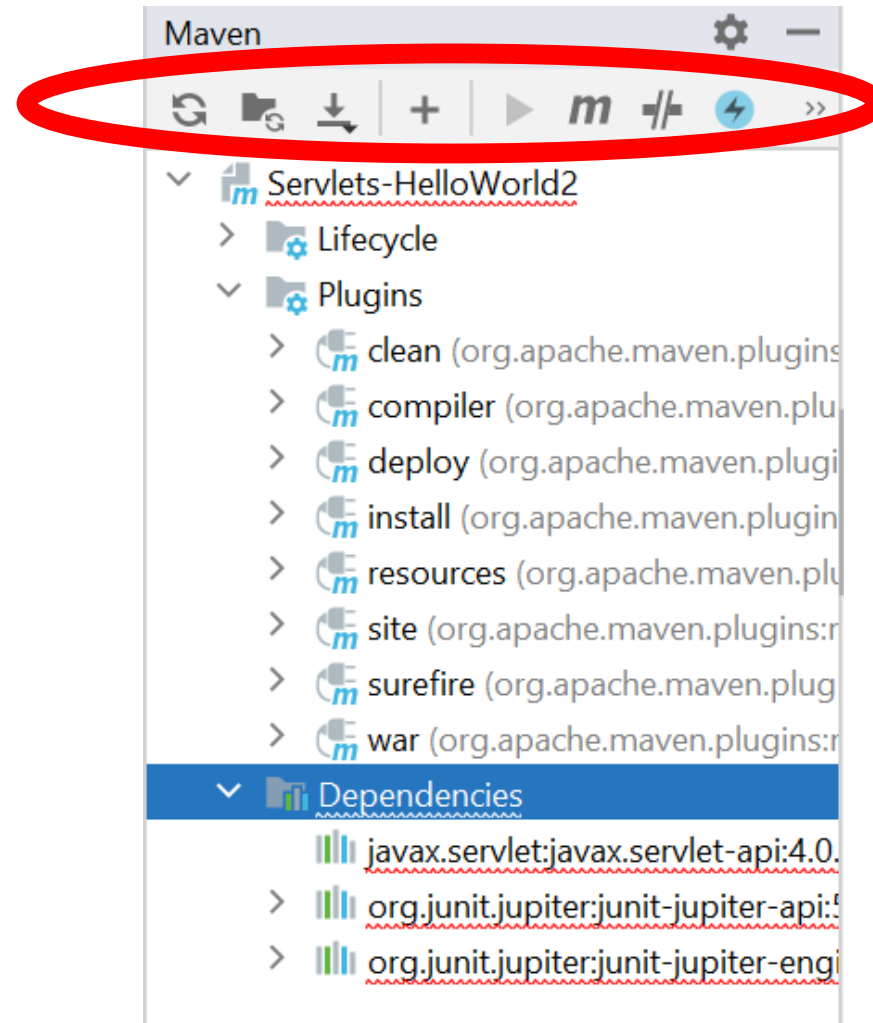
IntelliJ 2023 / Settings



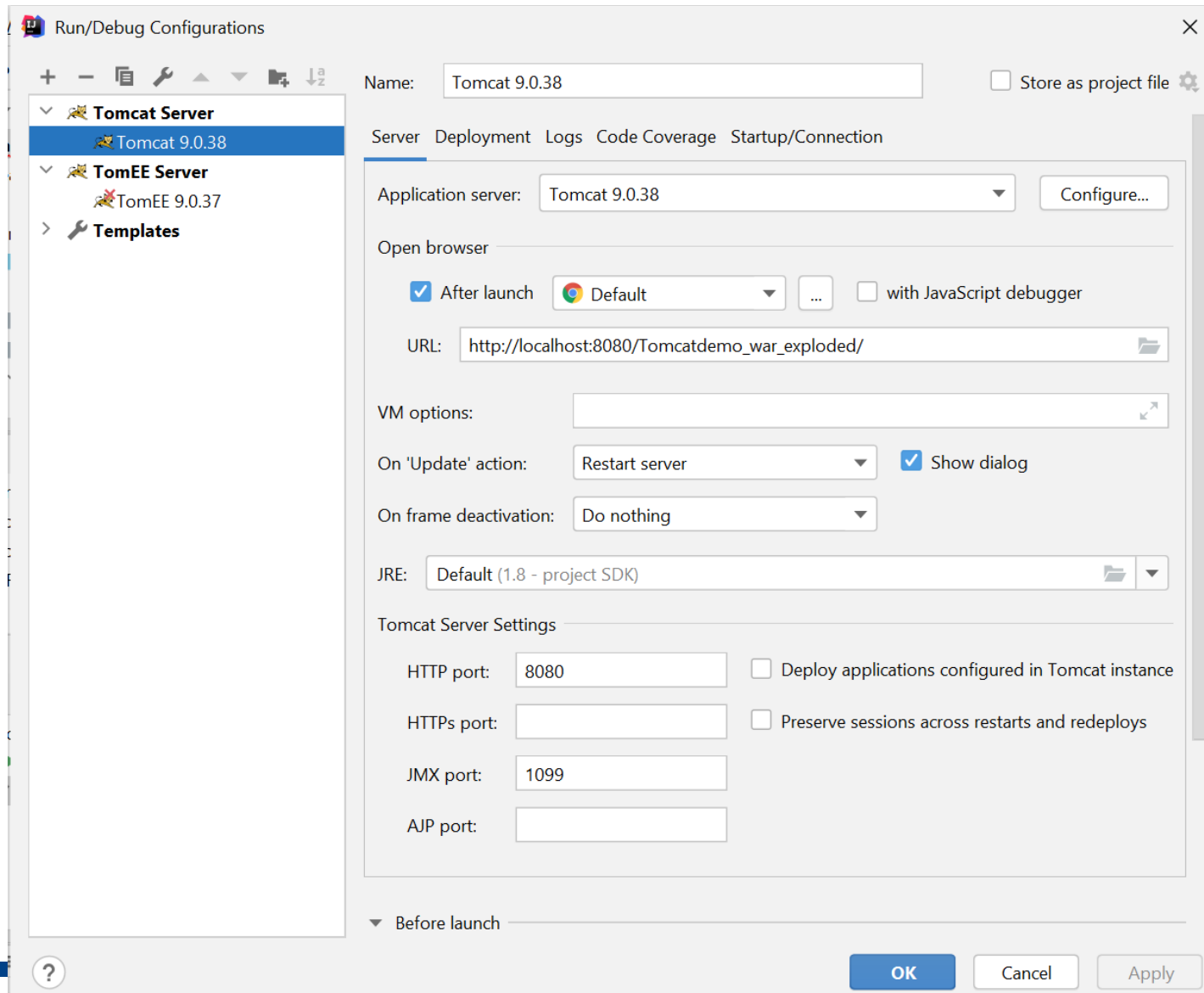
Tomcat 10 + IntelliJ 2023



Maven dependencies updaten!

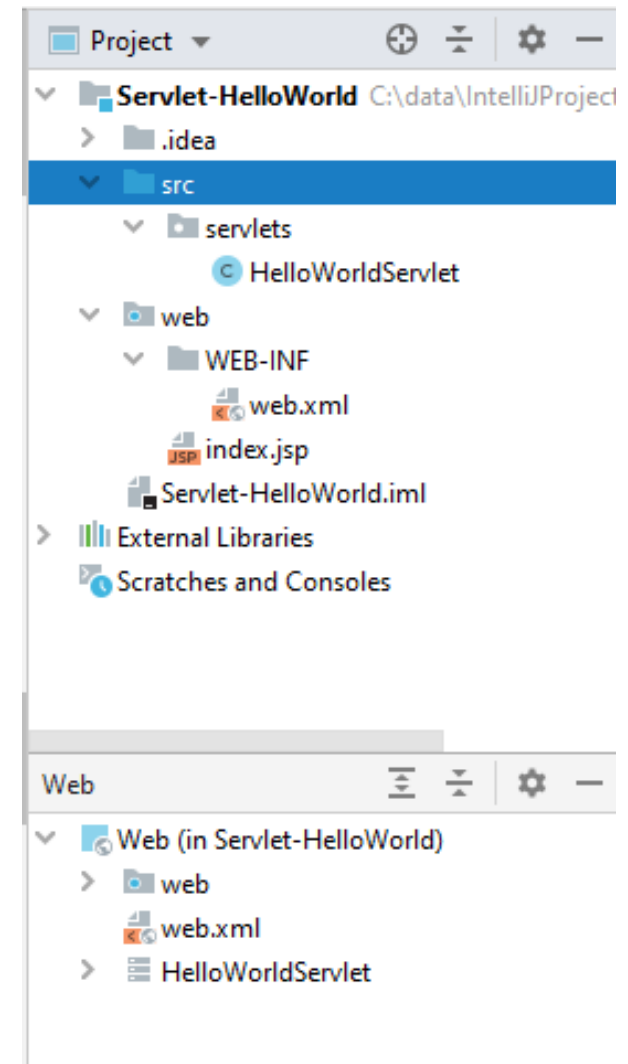


Edit Configurations / add artifact



Struktur einer Web-Applikation in IntelliJ

- **src**
 - Unkompilierter Java Code
- **web**
 - jsp, Java Script, HTML, images
 - Unterverzeichnisse
- **web/WEB-INF**
 - web.xml= "Deployment Descriptor"



- **Arbeite immer mit Custom (benutzerdefiniert, maßgeschneidert) URLs !**
- URLs haben mehr Bedeutung und sind einfacher, kürzer und stimmiger
- Web.xml kann für Initialisierung genutzt werden
- Filter und Sicherheitseinstellungen können verwendet werden

Mehr zu Custom URLs

- **Normal usage**
 - `<url-pattern>/blah</url-pattern>`
- Muss mit / beginnen
- Resultat URL
 - `http://somehost/someApp/blah`
- **Option: Verwendung von wildcards für:**
 - File extension (Anm: kein / in diesem Fall !)
 - `<url-pattern>*.html</url-pattern>`
 - Verzeichnisse
 - `<url-pattern>/dir1/dir2/*</url-pattern>`



Aufgabe

- Tomcat (Servlet Container) installieren!



- und dann geht's los mit einem Beispiel

Lebenszyklus eines Servlets

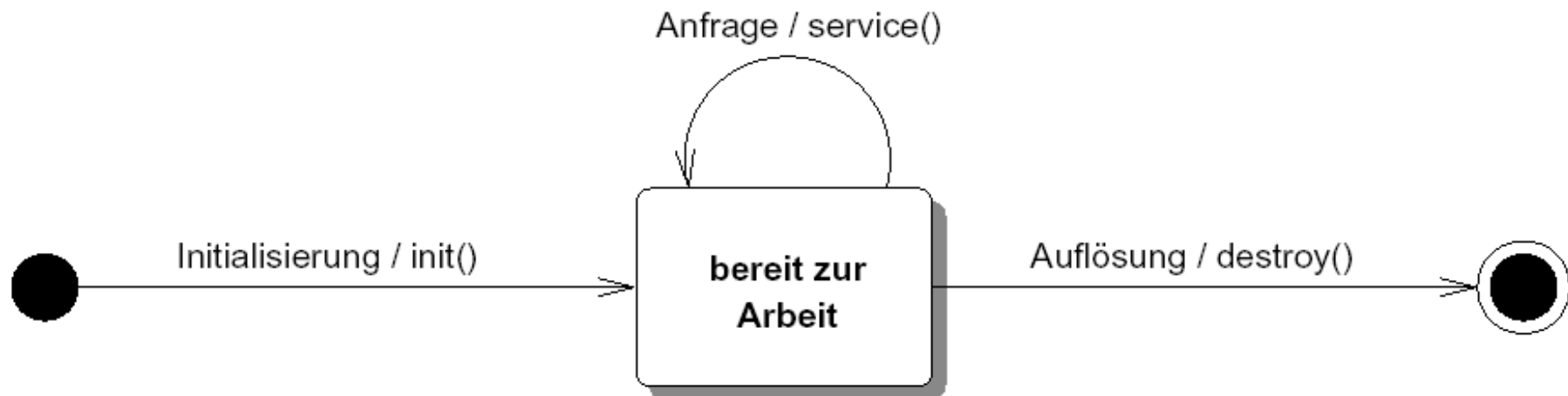
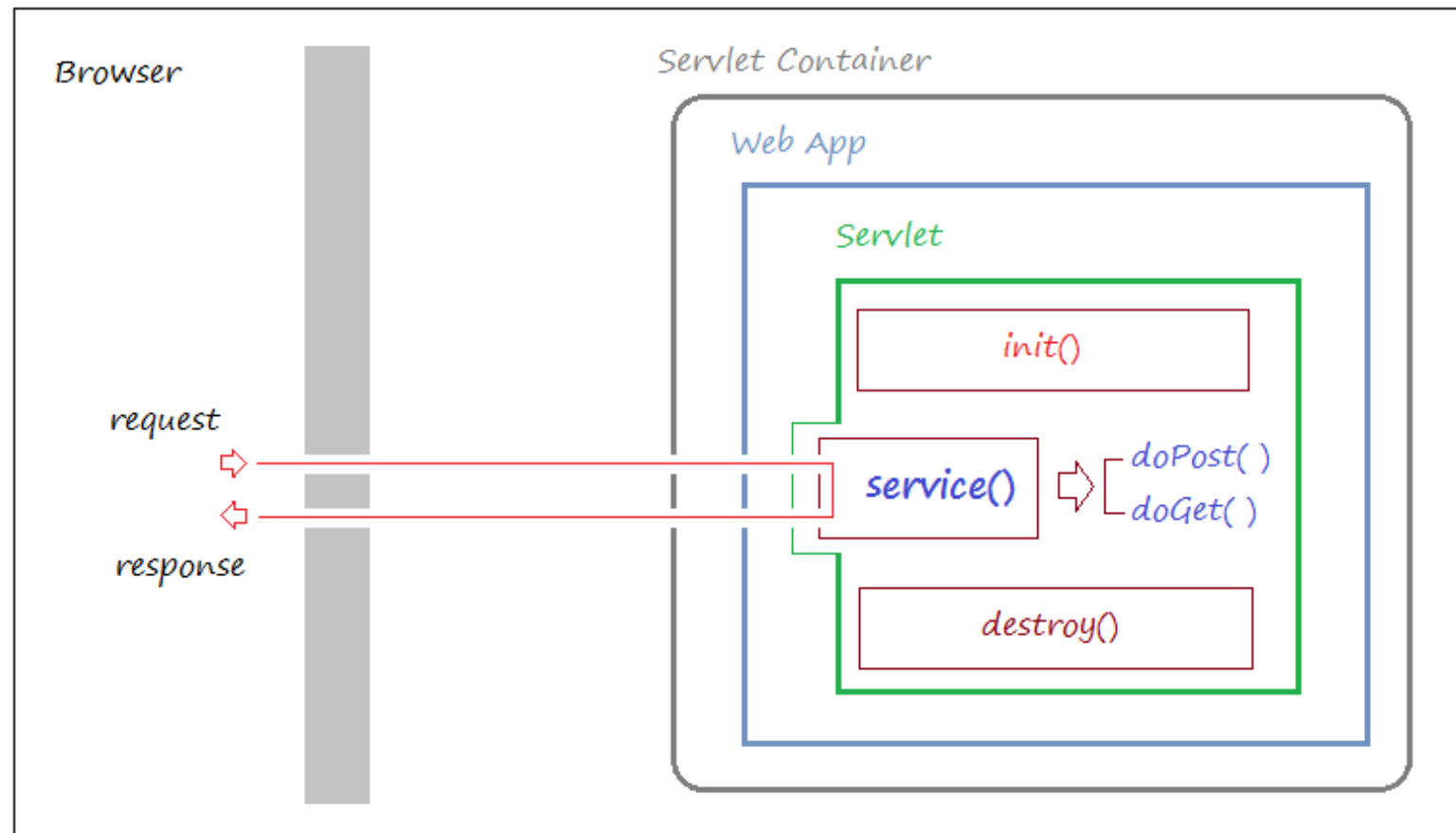


Abbildung 5.3: Lebenszyklus von Servlets

Lebenszyklus eines Servlets



Lebenszyklus eines Servlets (2)

- Der Lebenszyklus eines Servlets im Detail:
 - Laden und instanziiieren
 - Entweder beim Start des Servlet-Containers oder bei der ersten Anfrage
 - Initialisieren
 - Die *init*-Methode des Servlets wird aufgerufen
 - Hier kann das Servlet Initialisierungsaufgaben erledigen, z.B. eine Datenbankverbindung herstellen oder Konfigurationsdaten aus einer Datei oder aus dem Deployment Descriptor (web.xml) einlesen

Lebenszyklus eines Servlets (3)

- Client-Anfragen bearbeiten
 - Die *service*-Methode des Servlets wird aufgerufen (Soll nicht überschrieben werden)
 - Diese Methode überprüft den HTTP-Anfragetyp und leitet die Anfrage an die richtige Methode weiter, z.B. *doGet*, *doPost*
- Servlet-Klasse wieder entladen
 - Der Servlet Container entscheidet, wann die Servlet-Instanz wieder aus dem Speicher entfernt wird
 - Vorher wird die Methode *destroy* aufgerufen



Aufgaben

1. Hello World Servlet zum Laufen bringen

2. Tipp des Tages:

Schreibe ein Servlet, dass aus einer fixen Auswahl von Tipps (z.B.: Java, Spruch, Guten Rat, ...) zufällig einen auswählt und jedes Mal wenn ein Refresh gemacht wird, einen neuen zufällig ausgibt. Erstelle 8-10 Tipps.