

# Vorlesungsreihe

## Entwicklung webbasierter Anwendungen

# Webanwendungen mit Java

Prof. Dr.-Ing. Thomas Wiedemann  
email: [wiedem@informatik.htw-dresden.de](mailto:wiedem@informatik.htw-dresden.de)



HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)  
Fachbereich Informatik/Mathematik

- Allgemeine Optionen unter Java
  - Java-Historie und Haupteigenschaften
  - Überblick Client-Server-Anwendungen
- Webanwendungen mit Java
  - JavaServlets
  - Java Server Pages
  - Programmierbeispiele zu Servlets und Java Server Pages
  - Die Tomcat-Engine

# Historie von Java

- Anfang der 90er Jahre startete SUN das "Green-Projekt" zur zukünftigen Programmierung intelligenter Haushaltsgeräte (Toaster, Videorecorder, Fernseher, etc.)
- Bestandteile des Projekts war das Green-OS und ein portabler Interpreter (Oak)
- nach einem kommerziellen Mißerfolg des Greenprojektes wurde Oak als Interpretersprache innerhalb eines SUN-Browsers "Webrunner" verwendet
- die erfolgreiche Anwendung von Oak führte zur Umbenennung in HotJava
- die stabile Arbeitsweise von Hotjava führte zur Lizenzierung durch die Fa. Netscape im Dezember 1995 und wurde in Netscape 2.0 eingesetzt
- Anfang 1996 Freigabe des Java Development Kit - JDK 1.0
- sehr rascher Anstieg der Anwendungen
- Aktuelle Version - Java 8 (Stand Ende 2016) - Preview Java 9
- nach Übernahme von Sun durch Oracle tw. kommerzieller Support durch Oracle (akt. Version frei, Support älterer Versionen kostenpflichtig)
  - aktuell auch Gerichtsverfahren zw. Oracle und Google bzgl. Java-API's in Android ...

# Java im Überblick - grundlegende Sprachmerkmale

- Java wurde vollständig neu entworfen
- wesentliche Elemente wurden von C und C++ übernommen

## **Zielstellung der Entwickler :**

- »Java soll eine einfache, objektorientierte, verteilte, interpretierte, robuste, sichere, architekturneutrale, portable, performante, nebenläufige, dynamische Programmiersprache sein.«
- Ein Großteil dieser Ziele wurde gut umgesetzt !

## **Einige Vorteile von ergeben sich aus der Vermeidung von C/C++ - Problemen :**

- keine expliziten Pointer, keine separaten Header-Dateien
- bessere Laufzeitüberwachung bei Matrizen und anderen dynamischen Objekten
- Automatische Speicherverwaltung bei Strings und Objekten (Garbage Collector)
- keine Mehrfachvererbung und keine Templates in Java
- sehr gutes Fehlermanagement (wesentlich stabiler als C/C++)
- sehr gutes Sicherheitsmanagement (Vermeidung unsicherer Programmierpraktiken)
- portabel auf verschiedene Rechnerplattformen

## Realisierung als Interpretersprache

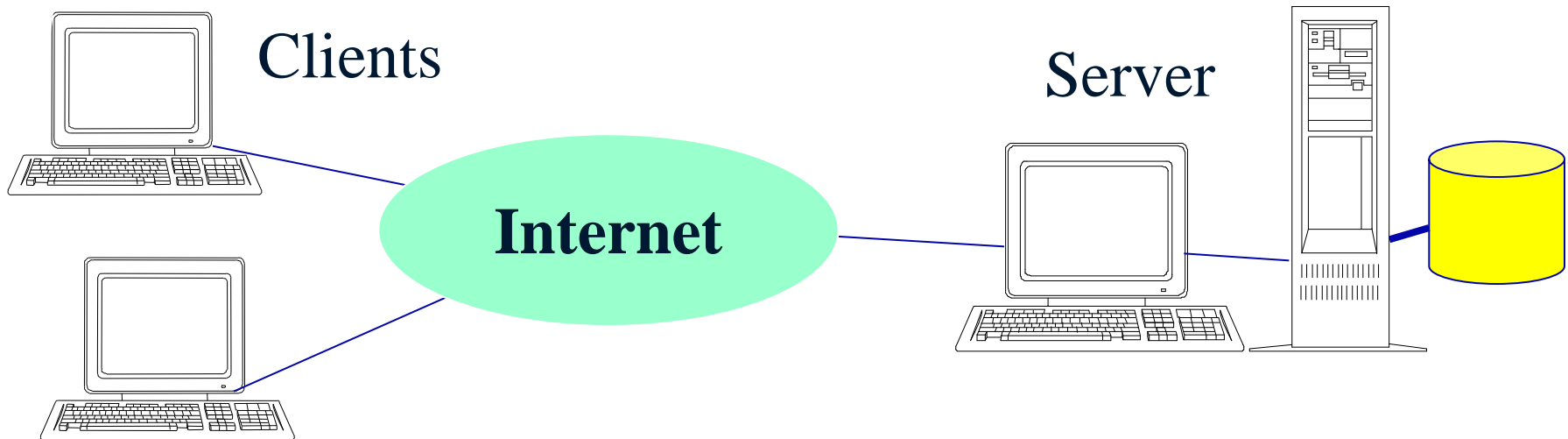
- Java-Quellcode wird übersetzt durch Java-Compiler ( javac ) in Bytecode
- Bytecode ist eine Art portabler Objektcode
- wird von einer Virtuellen Maschine (VM) interpretiert
- bei Bedarf kann Bytecode auch in nativen Code konvertiert werden (Just in Time Compiler)
- zukünftig sollen auch Prozessoren zur direkten Ausführung des Bytecodes verfügbar sein

## Sicherheitsmaßnahmen :

- Sprachentwurf vermeidet gefährliche Techniken (keine Pointer)
- innerhalb der VM wird eine in sich geschlossene Umgebung erzeugt (Sandbox) , in welcher nur eine beschränkte Anzahl von Funktionen verfügbar ist
- es werden keine direkten Zugriffe (d.h. ohne entsprechende VM-Funktionen) auf das Dateisystem des Rechners oder das Betriebssystem zugelassen

# Allgemeine Konfiguration von Webanwendungen

- Datenbank befindet sich generell auf dem Server, räumlich entfernt von den Clients



## Zwei Grundkonzepte bezüglich der Konfiguration

1. Konzentration der gesamten Anwendungslogik auf Server, Clientrechner realisiert nur sehr einfache Anzeige und Visualisierungsfunktionen (**Thin-Client-Konzept**)  
- unter Java mit **Java Servlets oder Java Server Pages**
2. Anwendungslogik wird auf dem Client realisiert (**Fat Client**), Server stellt nur Datenbankzugriff bereit -> **unter Java mit Applets (veraltet!)**
3. Mischformen sind möglich und sinnvoll – einfache Logik zwecks Plausibilitäts- und Fehlerkontrolle auf Client, komplexe Datenbankaktionen auf Server  
-> Thin-Fat-Client mit **HTML5/JavaScript und AJAX**

## Grundkonzepte

### A: Serverseitige Anwendung (Thin Client)

- Skriptlösung : Java Servlet in einem Servletcontainer wie Tomcat
- HTML-orientierte Lösung: Java Server Pages (siehe Vortrag)

### B. Clientbasierte Applikation (Fat Client)

- Laden eines Java-Applets in den Browser (**veraltet und in den Browsern nicht mehr unterstützt**)
- Java Applet baut ggf. Verbindung zum Server zwecks Datenbankabfrage auf

### C : Thin-Fat-Client mit Ajax

- - JavaScript kommuniziert per XML-Nachrichten mit Webserver und zeigt neue Inhalte ohne Neuladen der Seite an (Web 2.0)

- Servlets sind Java-Objekte, welche von `javax.servlet.Servlet` abgeleitet werden
- Servlets werden innerhalb eines Servlet-Containers (oder auch Servlet Engine) instanziiert und gestartet
- Häufigste Verwendung als `HTTPServlet`, das auf Browser-Anfragen (Request) antwortet
- Servlets bekommen vom Client (dem Browser) Parameter im Request-Objekt übergeben
- Servlets erzeugen HTML-Code, der als Response zum Client zurückgeliefert wird
- Servlet-Container sind Applikations-Server, die die J2EE Spezifikation erfüllen. Bekannte Hersteller hierfür sind Bea (WebLogic), IBM (WebSphere), Apache (Tomcat, JBoss, Open-Source), Enterprise Server (Borland), ...



# Java – Servlet – Beispiel

- Java-Programm zur Erzeugung von HTML-Ausgaben
- Beispiel erzeugt mit JBuilder-Assistenten für Java-Servlets :

```
public class Servlet1 extends HttpServlet { // HTTP-Servlet stellt alle Basisfunkt.
    private static final String CONTENT_TYPE = "text/html";
    /**Globale Variablen initialisieren*/
    public void init() throws ServletException {
    }
    /**Die HTTP-Anforderung Get bearbeiten*/
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Servlet1 </title></head>");
        out.println("<body>");
        out.println("<p>Das Servlet hat ein GET empfangen. Das ist die Antwort.</p>");
        out.println("</body></html>");
    }
}
```

- Alle Requestparameter werden durch `HttpServletRequest request` übergeben
- `Response` definiert `out.println()` - übergibt Texte an den Ausgabestream des Servers

## Initialisierung

- Methode void init () - Server lädt und erzeugt Instanz des Servlets

## Interaktion mit Clients (Behandlung von Anfragen)

Über Objekt HttpServlet

- void doGet (HttpServletRequest, HttpServletResponse)  
=> zur Behandlung von GET-Request (URL-Eingabe)
- void doPost (HttpServletRequest, HttpServletResponse)  
=> zur Behandlung von POST-Request (Formular-Actio-Button)
  
- void destroy () - Server entfernt Servlet-Instanz (i.d.R. erst beim Stoppen des Servers) => zum Aufräumen , z.B. zum Schliessen von Logfiles oder Datenverbindungen ....

Das Requestobjekt wird beim Aufruf über DOGET / DoPOst übergeben und enthält alle Informationen über den Aufruf von Seiten des Servers

Konkrete Methoden / Attribute :

- String getParameter (String) => liefert den Wert eines konkreten Parameters.
- Enumeration getParameterNames () - liefert die Namen sämtlicher Parameter, die im request- Objekt mitgeliefert wurden.
- String[] getParameterValues (String) - Diese Methode ist für Parameter geeignet, die mehr als einen Wert enthalten können. Zurückgeliefert wird ein ganzes Array von Werten des benannten Parameters.

# Java – Servlet- Formularverarbeitung

- **DoPost** verarbeitet Formulardaten aus `request.getParameters()`

```
/**Die HTTP-Anforderung Post bearbeiten*/
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType(CONTENT_TYPE); PrintWriter out =
response.getWriter();
    out.println("<html>"); out.println("<head><title>Servlet1</title></head>");
    out.println("<body>"); out.println("<p>POST - Das ist die Antwort.</p>");

    out.println("Parameterliste: ");
    Enumeration par;
    par = request.getParameterNames();
    String parname = (String)par.nextElement();
    out.println("<BR>P:"+parname);
    while ( par.hasMoreElements() )
    { out.println("<BR>P: "+(String)par.nextElement()); }
    String parvalue = request.getParameter(parname);

    out.println("<B><BR>P:"+parname + " = " + parvalue+"</B>");
    out.println("</body></html>");
}
```



## Probleme beim Einsatz von Servlets

- Vermischung von Präsentation und Verarbeitungslogik durch gleichzeitige Programmierung in EINEM Code
- **HTML nur als Print-Ausgabe, damit Designer-Werkzeuge für HTML nicht verwendbar !**

## Lösung : Java Server Pages (JSP)

- konzipiert für Web-Seiten mit dynamisch generiertem Inhalt
- Definition im HTML-Format mit eingebettetem Java-Code
- **wird zu Java-Servlet übersetzt !**

## Folgen :

- Striktere Trennung von fachlicher Logik und Präsentation
- Änderungen in GUI unabhängig von Verarbeitungslogik
- Verwendung wiederverwendbarer Komponenten, z.B. mit Komponenten (z.B. JavaBeans)

# Java-Server-Pages-Details

- Erstellung innerhalb eines statischen HTML-Dokumentes
- Einbettung von Java-Code über `<% ... %>`
- Deklarationen über `<%! ... %>`
- Die Objekte `out` und `request` stehen ohne Deklaration zur Verfügung.

## Versteckte Kommentare (Hidden Comment)

- Syntax: `<%-- Kommentar --%>`
- Zum Dokumentieren des Quellcodes - wird von der JSP-Engine nicht verarbeitet und wird somit auch nicht zum Client gesendet

## Ausgabe Kommentar (Output Comment)

- Syntax: `<!-- Kommentar [<%= Ausdruck %>] --!>`
- Wird von der JSP-Engine verarbeitet und das Ergebnis taucht in der gesendeten HTML-Quelldatei auf. Der im Kommentar enthaltene Ausdruck wird dynamisch zur Laufzeit bei jedem Aufruf evaluiert

# Java-Server-Pages-Beispiel

- Die Datei wird geparkt und der Java-Code wird übersetzt !

```
<html><head><title>JSPTest</title></head>
<h1>Spiegel Java Server Page</h1>
<body> <form method="post">
Eingabe: <input type="text" name="eingabe" />
<br>Ausgabe:
<%! // eine Deklaration
String eingabe, ausgabe; %>
<% // hier steht nun Java-Code ...
eingabe = request.getParameter("eingabe");
if(eingabe != null) {
ausgabe = ServletTest.reverse(eingabe);
out.println(ausgabe);
}
// Ende des Java-Codes
%>
<input type="submit" name="Submit" />
</form></body></html>
```



- Das Programmpaket TOMCAT ist eine Servlet-Engine.

## **Diese Engine :**

- verwaltet das Starten und die Parameterübergabe zwischen Webserver und Servlet.
- Bei einem Aufruf des Servlets wird dieses durch die TOMCAT-Engine aktiviert und bleibt anschließend aktiv.
- Die TOMCAT-Engine kann auch ein Servlet mehrfach ausführen und verfügt dazu über Multitasking-Fähigkeit

## **Tomcat kann in drei Modi arbeiten**

- als einfacher Webserver (siehe Übung) auch mit eigener SSL-Verschlüsselung
- integriert in andere Webserver (allgemeiner Modul)
- Speziell angebunden an Apache mit Zusatzfunktionen



# Tomcat in der Apache-Gesamtkonfiguration

- TOMCAT als Servlet-Engine an einem Apache-Server

