

# Webdevelopment (in Java)





# Übersicht

- Meta-Programming
- Reflection
  - instanceof Operator
  - Annotations
- Webservices
- Spring



# instanceof-Operator

- vergleicht ein Objekt mit einem bestimmten Typ
- testet ob ein Objekt ist
  - eine Instanz einer Klasse
  - eine Instanz einer Unterklasse
  - eine Instanz einer Klasse, die eine Interface implementiert
- null ist nicht eine Instanz einer Klasse



# instanceof-Operator

```
class Parent {}  
class Child extends Parent implements MyInterface {}  
interface MyInterface {}
```

```
Parent obj1 = new Parent();  
Parent obj2 = new Child();
```

obj1 instanceof Parent	→ true
obj1 instanceof Child	→ false
obj1 instanceof MyInterface	→ false
obj2 instanceof Parent	→ true
obj2 instanceof Child	→ true
obj2 instanceof MyInterface	→ true



# instanceof-Operator

```
interface Printable{}
```

```
class A implements Printable{  
    public void a(){System.out.println("a method");}  
}
```

```
class B implements Printable{  
    public void b(){System.out.println("b method");}  
}
```

```
class Call{  
    void invoke(Printable p){  
        if(p instanceof A){  
            A a=(A)p;//Downcasting  
            a.a();  
        }  
    }  
}
```

```
        if(p instanceof B){  
            B b=(B)p;//Downcasting  
            b.b();  
        }  
    }  
}
```



# Annotations

- Art von Metadata
- bietet über das Programm Information, die nicht direkt zum Programm gehört
- haben keine direkte Auswirkung auf die Ausführung des Codes



# Annotations

- Information für den Compiler mit dem Ziel
  - Errors zu entdecken
  - Warnings aufzuheben
- Compile-Time Verarbeitung
  - Werkzeuge können Annotations nutzen um Code, XML Dateien usw. zu erzeugen
- Laufzeit Verarbeitung
  - manche Annotations sind am Laufzeit geprüft



# Annotations

- @ zeigt dem Compiler, dass der Ausdruck eine Annotation deklariert
  - @Entity
  - @Id
  - @SuppressWarnings
- Annotations können Elemente enthalten
  - @Entity(tableName = "vehicles", primaryKey = "id")





# Annotations

@Entity

```
public class Vehicle {
```

```
    @Persistent
```

```
    protected String vehicleName = null;
```

@Getter

```
    public String getVehicleName() {
```

```
        return this.vehicleName;
```

```
    }
```

```
    public void setVehicleName(@Optional vehicleName) {
```

```
        ...
```

```
    }
```

# Vordefinierte Annotations

@Deprecated zeigt dass das Element soll nicht mehr benutzt sein

```
// Javadoc comment follows
```

```
/**  
 * @Deprecated  
 * explanation of why it was deprecated  
 */
```

```
@Deprecated  
static void deprecatedMethod() { }
```



## Vordefinierte Annotations

@Override zeigt dass das Element ein Element von der  
Bassisklasse überschreibt

```
// mark method as a superclass method
```

```
// that has been overridden
```

```
@Override
```

```
int overriddenMethod() {}
```



## Vordefinierte Annotations

@SuppressWarnings zeigt den Compiler Warnings aufzuheben

```
// use a deprecated method and tell  
// compiler not to generate a warning
```

```
@SuppressWarnings("deprecation")  
void useDeprecatedMethod() {  
    // deprecation warning  
    // - suppressed  
    objectOne.deprecatedMethod();  
}
```



# Benutzerdefinierte Annotations

- Benutzerdefinierten Annotations sind erlaubt
- Annotations sind in ihren eigenen Datei definiert

```
@interface MyAnnotation {  
    String value();  
    String name();  
    int age();  
    String[] newNames();  
}
```



# Benutzerdefinierte Annotations

```
@MyAnnotation(  
value="123",  
name="ABC",  
age=0,  
newNames={"AAA", "BBB"}  
)
```

```
public class MyClass {  
...  
}
```



# Was genau ist eine Klasse?

- Eine Sammlung von
  - Felder
  - Methoden
  - Konstruktoren
- Durch Reflection werden wir solche Dinge nicht definieren sondern zugreifen



# Java Reflection

- Bestimmen die Klasse eines Objektes
- Daten über eine Klasse
  - Zugriffsmodifikatoren
  - Basisklasse
  - Eigenschaften
  - Methoden
- Erlaubt
  - Instanzen erzeugen
  - Methoden Aufrufen
  - Eigenschaften verändern





# Programming vs Reflecting

- Wir werden Reflection nutzen, um Objekte, die schon existieren, zu ändern
- Wir können dynamisch den Zustand eines Objektes, egal wie die Objekte implementiert sind
- Im Endeffekt können wir Objekte zur Laufzeit ändern
- Programme haben Zugriff auf strukturelle Repräsentation ihrer selbst

# Laufzeit

- Der typische Ablauf wäre
  - Klassen, Methoden umsetzen
  - Compile
  - Ausführung
- Änderungen erfordern den Code nochmal zu compilieren und auszuführen



# Reflection

- Typische Anwendungsfälle
- Erstellen von Objekten der Klassen, deren Typ unbekannt zur Kompilierzeit sind
- Dynamische Objekterzeugung + Manipulation + Methodenaufrufe
- Testing
- Debugging



## java.lang.reflect.\*

- Method
  - beschreibt die Methoden einer Klasse und erlaubt Zugriff
- Field
  - beschreibt die Felder einer Klasse: Name, Typ, etc.
- Constructor<T>
  - schafft Info über die Konstruktoren und erlaubt die Ausführung eines Konstruktor, um Instanzen zu erzeugen



# Wie beginnen wir?

- `java.lang.Class`
  - der Bauplan der Klasse
- `Class<? extends Object> theClass = ClassName.class;`
- `Class theClass = Class.forName("package.class");`
- bewirkt Laden des zugehörigen class-Files durch ClassLoader



# Felder

- `getFields()`
  - Alle public Felder in der Klasse (und auch von den Basisklassen)
- `getDeclaredFields()`
  - Gibt ein Array mit allen Felder zurück
- `getField(String name)`
  - Gibt das Feld mit einem bestimmten Name zurück



# Felder

- `get(Object obj)`
  - Gibt den Wert des Feldes zurück
- `getPrimitiveType(Object obj)`
- `set(Object obj, Object value)`
  - Verändert den Wert des Feldes
- `setPrimitiveType(Object obj, PrimitiveType value)`
- `getType()`
  - Gibt den Typ des Feldes zurück
- `getName()`
  - Gibt den Name zurück



# Methoden

- `getMethods()`
  - Alle public Methoden in der Klasse (und auch von den Basisklassen)
- `getDeclaredMethods()`
  - Gibt alle Methoden zurück
- `getName()`
  - Gibt den Name der Methode zurück
- `getReturnType()`
  - Gibt den Typ des Rückgabewertes der Methode zurück
- `getParameterTypes()`
  - Gibt den Typ der Parameters zurück
- `invoke(Object obj, Object... args)`
  - Ruft die Methode mit bestimmten Parameters auf





## Aufruf einer Methode

```
getMethod(String name, Class<?>... parameterTypes);  
  
public int doSomething(String stuff, int times, int max){}  
  
getMethod("doSomething", String.class, int.class, int.class);
```



# Konstruktoren

`getConstructors()`

`getDeclaredConstructors()`

`getConstructor(Class<?>... parameterTypes);`



# AccessibleObject

- `isAccessible()`
  - public Methoden, Felder, Konstruktoren geben `True` zurück
  - alle andere geben `False` zurück
- `setAccessible(boolean flag)`



## Beispiel

```
import java.lang.annotation.* ;

import java.lang.reflect.* ;

@Retention(RetentionPolicy.RUNTIME)
@interface Copyright {
    String author();
    int year();
}
```



## Beispiel

```
@Copyright(author="cat", year=2018)
class T {
    public void f () {System.out.println("f() in T");}

    private void f (String x) {System.out.println(x+"->f() in
T");}
    private String x;

    public T (String x) { this.x = x;}
}
```



## Beispiel

```
public class main {  
    public static void main (String[] arg) throws Exception {  
        Class cls = null;  
        cls = Class.forName( "T" ) ;  
  
        Copyright cr = (Copyright) cls.getAnnotation(  
Copyright.class) ;  
  
        System.out.println(cr.author());  
        System.out.println(cr.year());  
        Constructor c = cls.getConstructor(String.class);  
        Object clsi = c.newInstance("aaa");  
    }  
}
```

## Beispiel

```
//Object clsi = cls.newInstance();  
//for (Method m : cls.getMethods())
```

```
Method m = cls.getMethod("f");  
m.invoke(clsi);  
m = cls.getDeclaredMethod("f", String.class);  
m.setAccessible(true);  
m.invoke(clsi, "10");
```

```
Field f = cls.getDeclaredField("x");  
f.setAccessible(true);  
System.out.println(f.get(clsi));
```

```
}
```

```
}
```



The HTML Editor



CoffeeCup







# Welcome to Amazon.com Books!

One million titles,  
consistently low prices.

(If you explore just one thing, make it our personal notification service. We think it's very co

## SPOTLIGHT! -- AUGUST 16TH

These are the books we love, offered at Amazon.com low prices. The spotlight moves EVE  
day so please come often.

## ONE MILLION TITLES

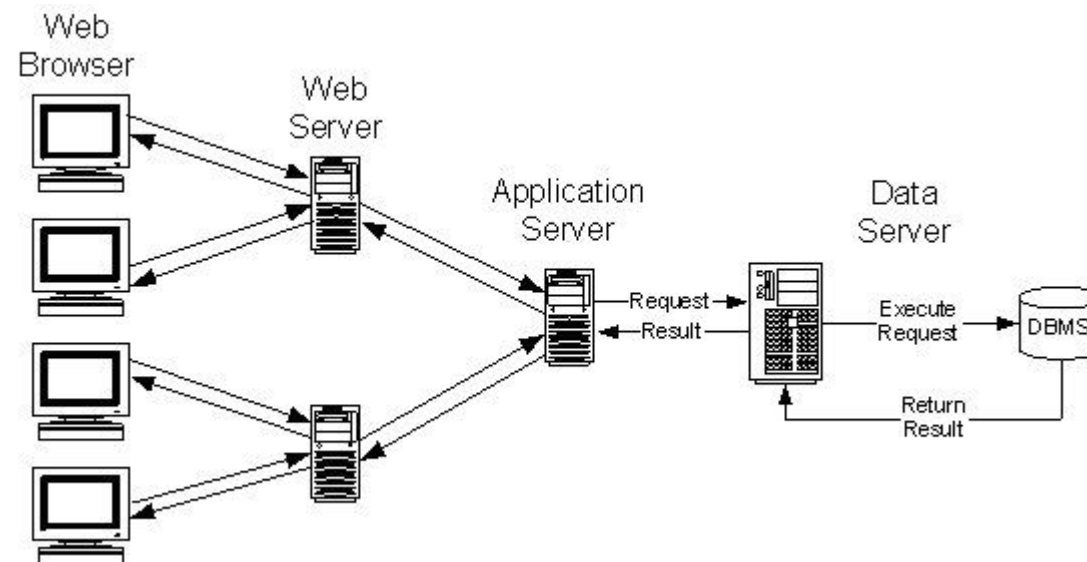
Search Amazon.com's [million title catalog](#) by author, subject, title, keyword, and more... Or  
a look at the [books we recommend](#) in over 20 categories... Check out our [customer review](#)  
the [award winners](#) from the Hugo and Nebula to the Pulitzer and Nobel.. and [bestsellers](#) ar  
30% off the publishers list..





# Client-Server-Modell

- Asymmetrisches Modell
- Server stellen Dienste bereit, die von Clients genutzt werden können





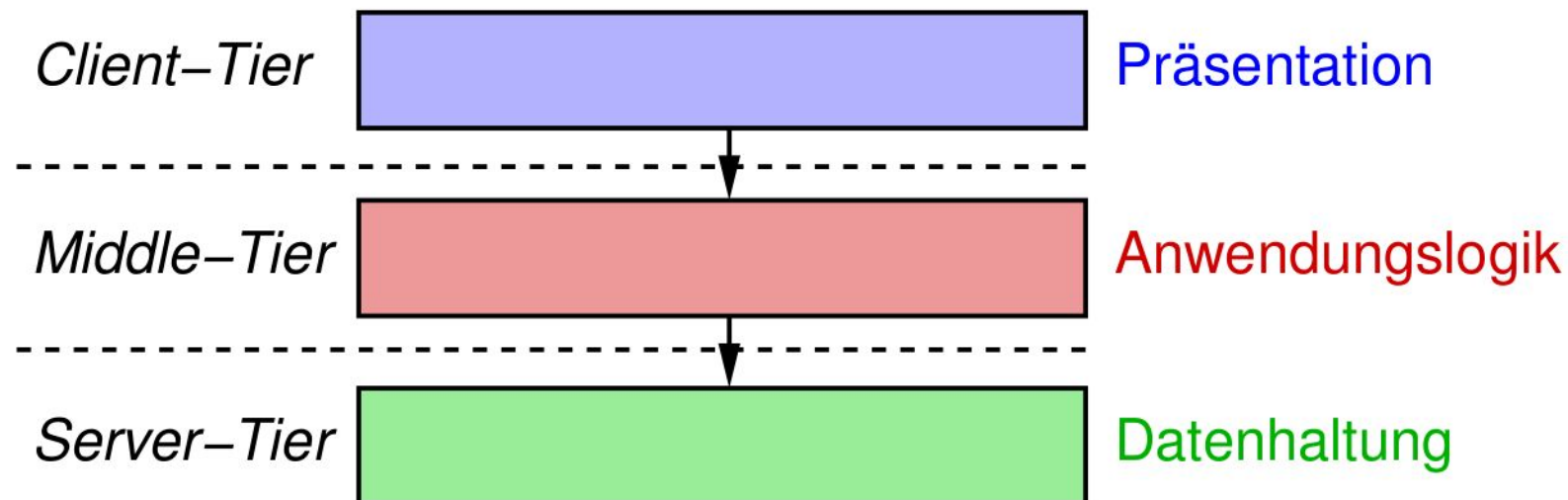
# n-Tier-Architektur

- Verfeinerungen der Client/Server-Architektur
- Modelle zur Verteilung einer Anwendung auf die Knoten einer verteilten Systems
- Vor allem bei Informationssystemen verwendet
- Tier kennzeichnet einen unabhängigen Prozessraum innerhalb einer verteilten Anwendung

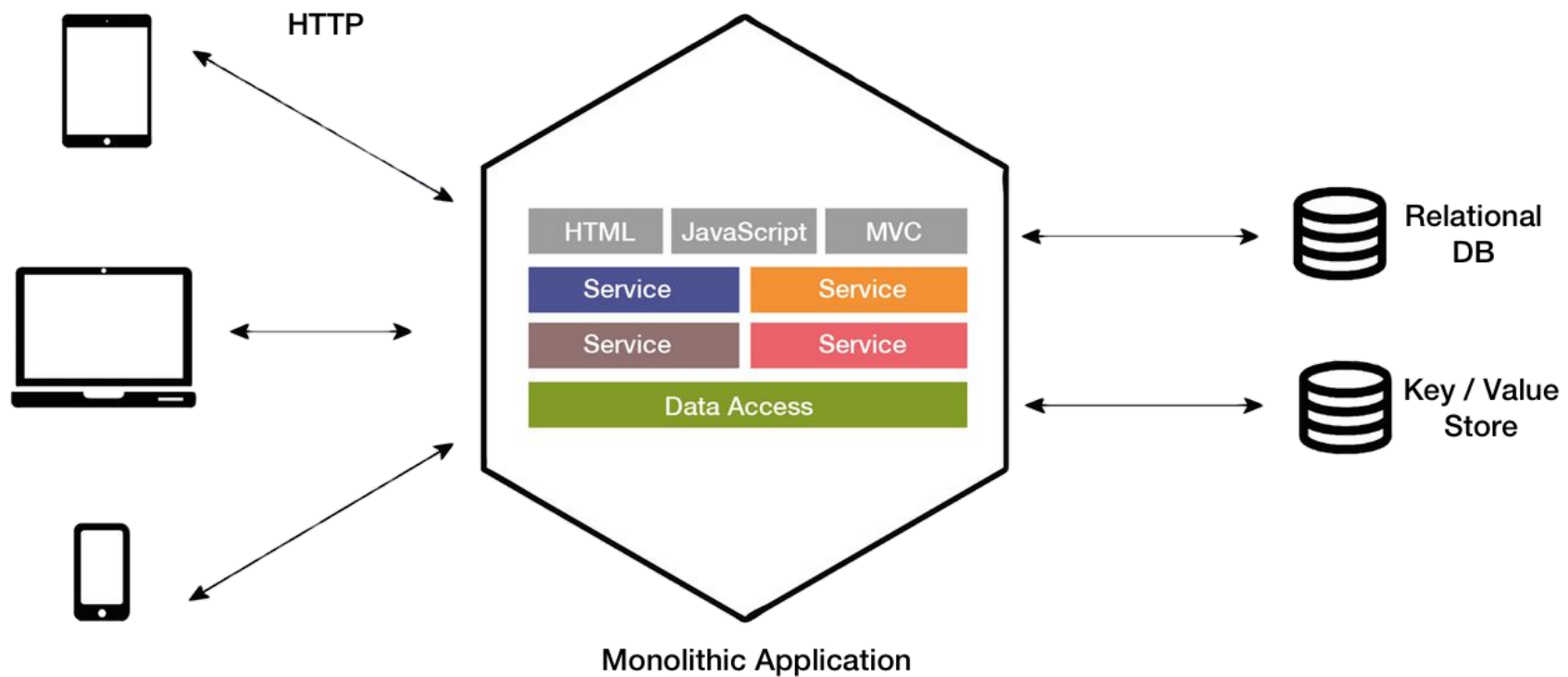


## 3-Tier-Architektur

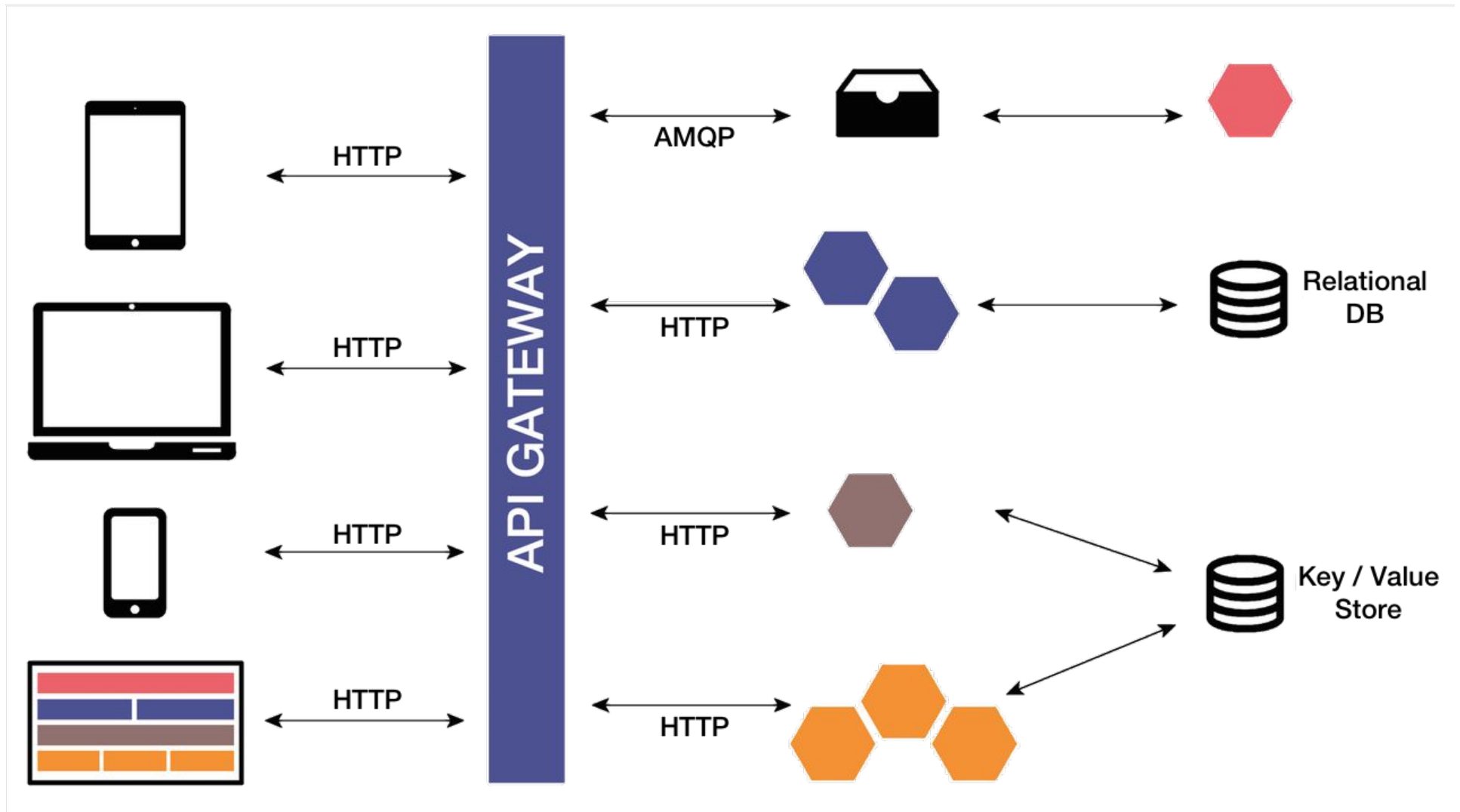
- Standard-Verteilungsmodell für einfache Web-Anwendungen
- Client-Tier : Web-Browser zur Anzeige
- Middle-Tier : Web-Server mit SpringBoot, Cloud Native, etc.
- Server-Tier : Datenbank-Server



# Typische Anwendung



# Typische Anwendung





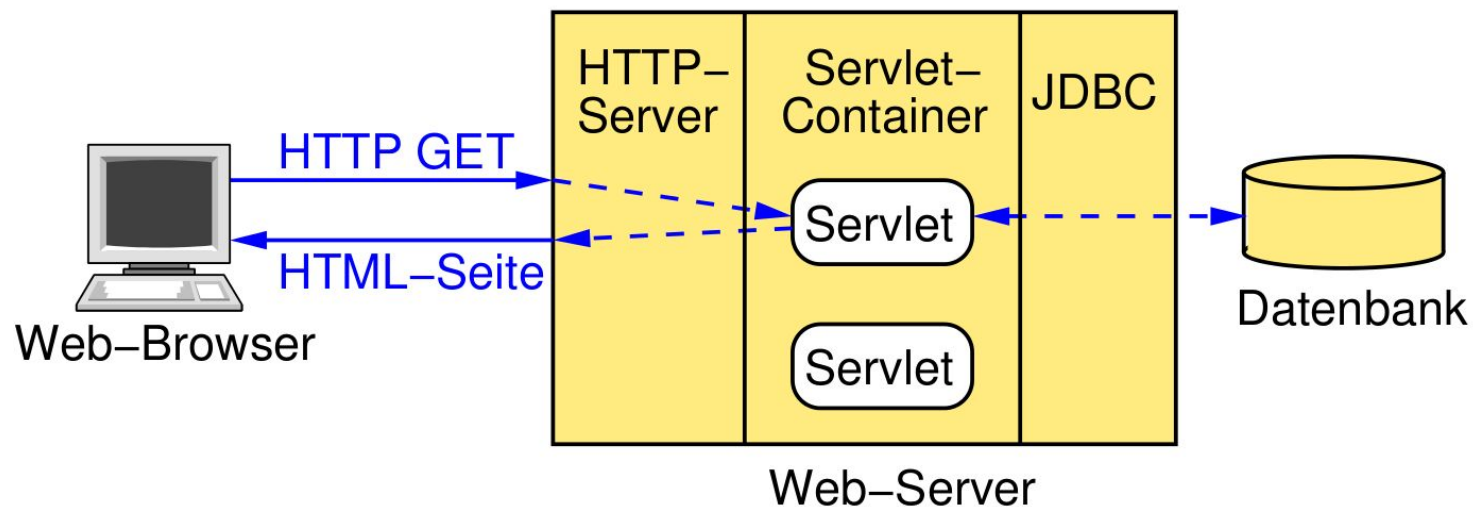
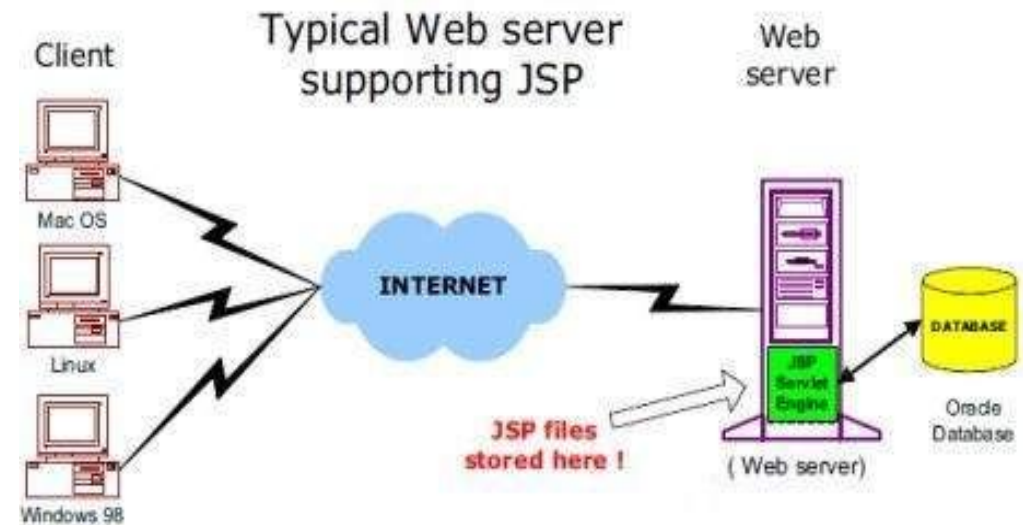
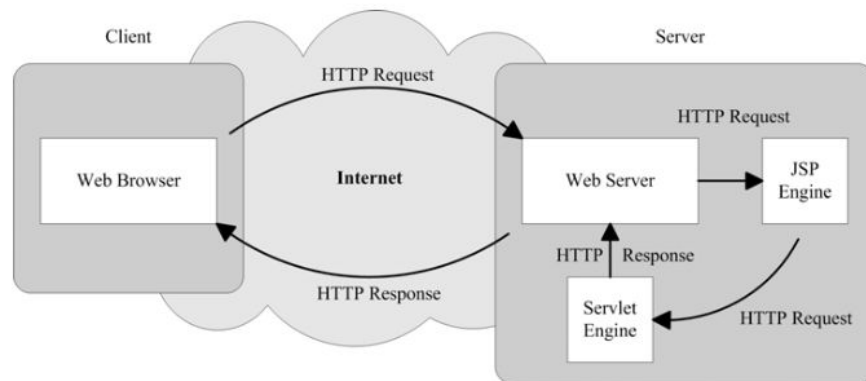


# Grundlagen

- Implementation von Java-Klassen, die innerhalb eines Web-Servers ausgeführt werden
- Web-Server muß servlet-fähig sein, d.h. über einen Servlet-Container verfügen
- Server wird über HTTP-Anfragen angesprochen (GET, POST)
- Server bearbeitet die Anfrage und erzeugt eine HTML-Seite bzw. ein Response
- Bearbeitung erfolgt durch eigenen Thread im Adressraum des Web-Servers

# Alte Welt/Servlets

- Servlets
- Java Server Pages (JSP)



# Hear from the Spring team this January at SpringOne

Virtual • Free • Jan 24–26

[REGISTER NOW](#)

**NEWS**

[Native Support in Spring Boot 3.0.0-M5](#)

## What Spring can do



### Microservices

Quickly deliver production-grade features with independently evolvable microservices.



### Reactive

Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.



### Cloud

Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.



### Web apps

Frameworks for fast, secure, and responsive web applications connected to any data store.



### Serverless

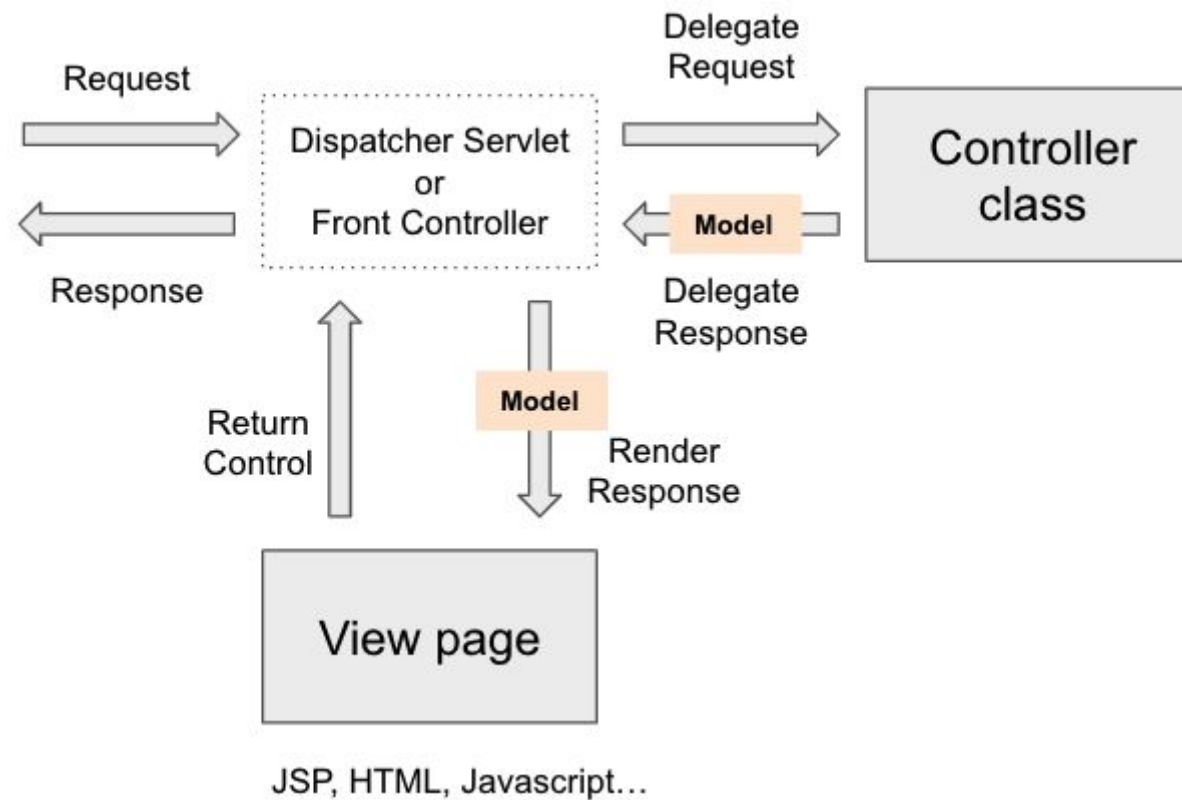


### Event Driven



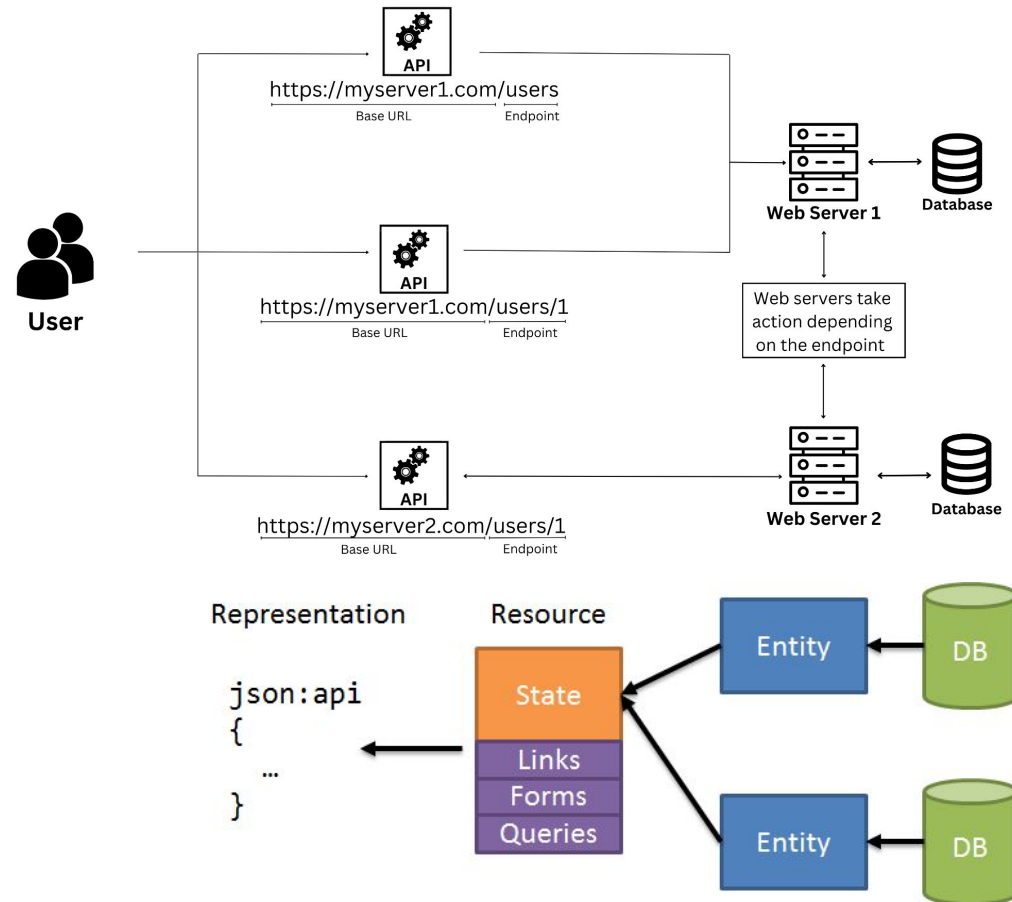
### Batch

# Spring - Model View Controller



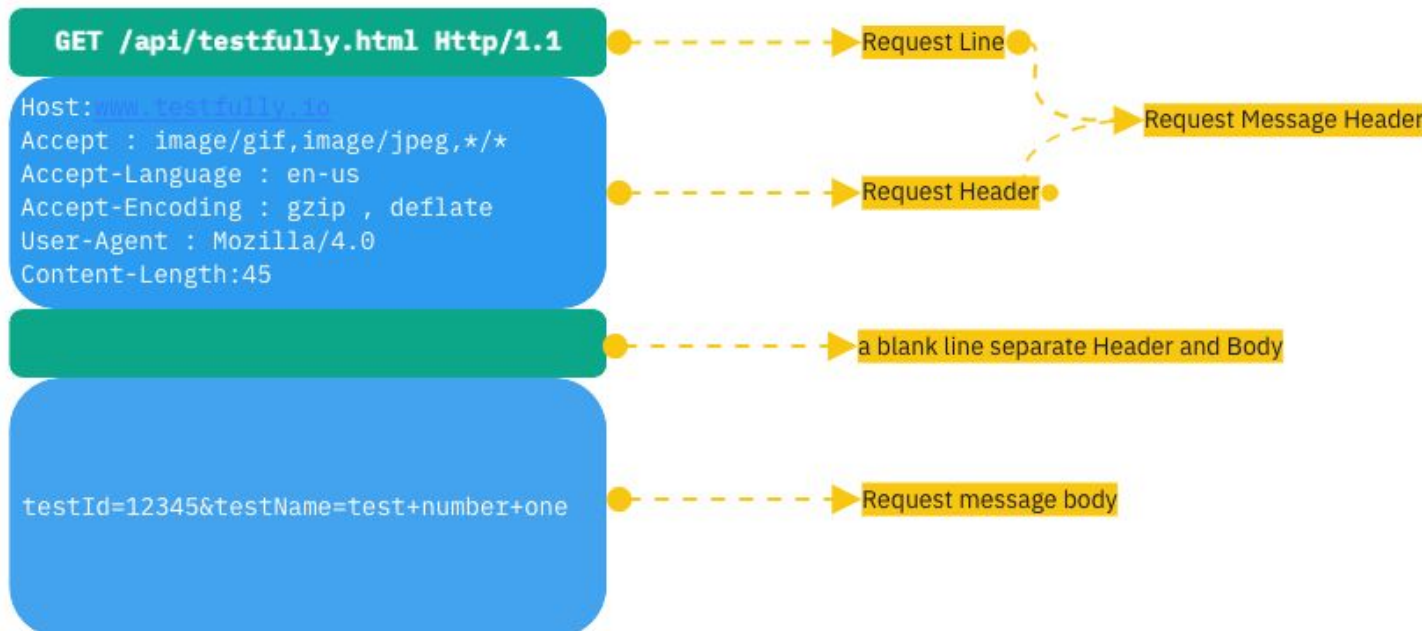
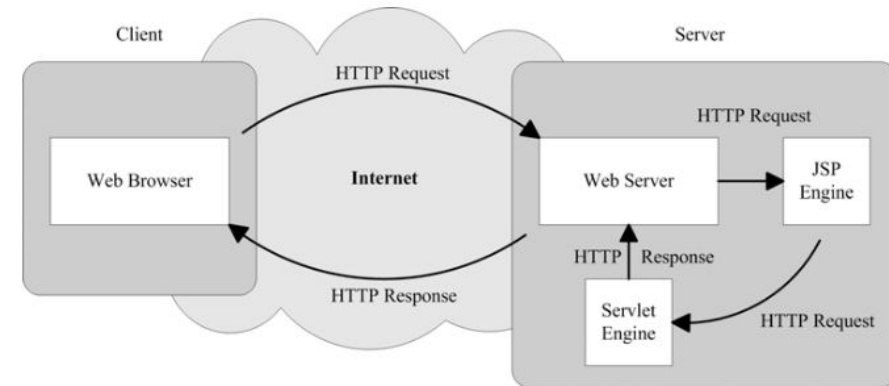
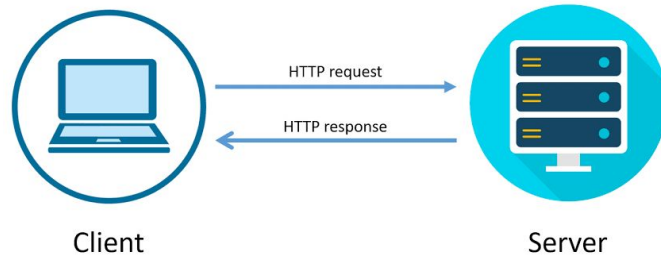
# REST

- REST ist ein Architekturstil
- läuft über HTTP
  - HEADERS und HTTP Verbs
- kein Standard
- RESTful
- viele Empfehlungen
- [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)



# REST

## Was ist ein Request?





# HTTP Verbs

- Teil des HTTP-Protokolls: Browser-Anfragen an den Server
- Auch verwendet in HTML-Formularen
- GET-Methode
  - zum Holen von Dokumenten über eine URL bestimmt
  - URL kann auch weitere Parameter beinhalten
  - GET /buy.html?what=shoe&price=50.00 HTTP 1.0
  - begrenzte Länge der URL
- POST-Methode
  - zum Senden von Daten an den Web-Server
  - Parameter werden im Rumpf der HTTP-Anfrage übertragen, sind in der URL nicht sichtbar
- wird später detailliert



# HTTP Verbes

- GET
  - information retrieval
- POST
  - Ressourcen erzeugen (CREATE, UPDATE)
- DELETE
  - Ressourcen löschen
- PUT
  - Ressourcen erzeugen (CREATE, UPDATE). URI
  - d.h. ID ist angegeben



# Status Codes

• geben die Antwort eines Servers auf ein Request an

• Kategorien:

- 1xx - Information
- 2xx - Success
- 3xx - Redirection
- 4xx - Request Error
- 5xx - Server Error

<b>Informational Status Codes</b> <b>100 – Continue</b> [The server is ready to receive the rest of the request.] <b>101 – Switching Protocols</b> [Client specifies that the server should use a certain protocol and the server will give this response when it is ready to switch.] <b>Client Request Successful</b> <b>200 – OK</b> [Success! This is what you want.] <b>201 – Created</b> [Successfully created the URI specified by the client.] <b>202 – Accepted</b> [Accepted for processing but the server has not finished processing it.] <b>203 – Non-Authoritative Information</b> [Information in the response header did not originate from this server. Copied from another server.] <b>204 – No Content</b> [Request is complete without any information being sent back in the response.] <b>205 – Reset Content</b> [Client should reset the current document. I.e. A form with existing values.] <b>206 – Partial Content</b> [Server has fulfilled the partial GET request for the resource. In response to a Range request from the client. Or if someone hits stop.] <b>Request Redirected</b> <b>300 – Multiple Choices</b> [Requested resource corresponds to a set of documents. Server sends information about each one and a URL to request them from so that the client can choose.] <b>301 – Moved Permanently</b> [Requested resource does not exist on the server. A Location header is sent to the client to redirect it to the new URL. Client continues to use the new URL in future requests.] <b>302 – Moved Temporarily</b> [Requested resource has temporarily moved. A Location header is sent to the client to redirect it to the new URL. Client continues to use the old URL in future requests.] <b>303 – See Other</b> [The requested resource can be found in a different location indicated by the Location header, and the client should use the GET method to retrieve it.] <b>304 – Not Modified</b> [Used to respond to the If-Modified-Since request header. Indicates that the requested document has not been modified since the specified date, and the client should use a cached copy.] <b>305 – Use Proxy</b> [The client should use a proxy, specified by the Location header, to retrieve the URL.] <b>307 – Temporary Redirect</b> [The requested resource has been temporarily redirected to a different location. A Location header is sent to redirect the client to the new URL. The client continues to use the old URL in future requests.]	<b>Client Request Incomplete</b> <b>400 – Bad Request</b> [The server detected a syntax error in the client's request.] <b>401 – Unauthorized</b> [The request requires user authentication. The server sends the WWW-Authenticate header to indicate the authentication type and realm for the requested resource.] <b>402 – Payment Required</b> [reserved for future.] <b>403 – Forbidden</b> [Access to the requested resource is forbidden. The request should not be repeated by the client.] <b>404 – Not Found</b> [The requested document does not exist on the server.] <b>405 – Method Not Allowed</b> [The request method used by the client is unacceptable. The server sends the Allow header stating what methods are acceptable to access the requested resource.] <b>406 – Not Acceptable</b> [The requested resource is not available in a format that the client can accept, based on the accept headers received by the server. If the request was not a HEAD request, the server can send Content-Language, Content-Encoding and Content-Type headers to indicate which formats are available.] <b>407 – Proxy Authentication Required</b> [Unauthorized access request to a proxy server. The client must first authenticate itself with the proxy. The server sends the Proxy-Authenticate header indicating the authentication scheme and realm for the requested resource.] <b>408 – Request Time-Out</b> [The client has failed to complete its request within the request timeout period used by the server. However, the client can re-request.] <b>409 – Conflict</b> [The client request conflicts with another request. The server can add information about the type of conflict along with the status code.] <b>410 – Gone</b> [The requested resource is permanently gone from the server.] <b>411 – Length Required</b> [The client must supply a Content-Length header in its request.] <b>412 – Precondition Failed</b> [When a client sends a request with one or more If... headers, the server uses this code to indicate that one or more of the conditions specified in these headers is FALSE.] <b>413 – Request Entity Too Large</b> [The server refuses to process the request because its message body is too large. The server can close connection to stop the client from continuing the request.] <b>414 – Request-URI Too Long</b> [The server refuses to process the request, because the specified URI is too long.] <b>415 – Unsupported Media Type</b> [The server refuses to process the request, because it does not support the message body's format.] <b>417 – Expectation Failed</b> [The server failed to meet the requirements of the Expect request-header.]	<b>Server Errors</b> <b>500 – Internal Server Error</b> [A server configuration setting or an external program has caused an error.] <b>501 – Not Implemented</b> [The server does not support the functionality required to fulfill the request.] <b>502 – Bad Gateway</b> [The server encountered an invalid response from an upstream server or proxy.] <b>503 – Service Unavailable</b> [The service is temporarily unavailable. The server can send a Retry-After header to indicate when the service may become available again.] <b>504 – Gateway Time-Out</b> [The gateway or proxy has timed out.] <b>505 – HTTP Version Not Supported</b> [The version of HTTP used by the client is not supported.] <b>Unused status codes</b> <b>306 – Switch Proxy</b> <b>416 – Requested range not satisfiable</b> <b>506 – Redirection failed</b>
---	---	--

## HTTP protocol version 1.1 Server Response Codes

<http://www.w3.org/Protocols/rfc2616/rfc2616.html>

Chart created September 5, 2000 by Suso Banderas(suso@suso.org). Most of the summary information was gathered from Appendix A of "Apache Server Administrator's Handbook" by Mohammed J. Kabir.

# REST

- Clients und Servers müssen sich durch eine Verhandlung auf einen gemeinsamen Darstellungstyp einigen
- Client gibt durch den **Accept-Header** an, was benötigt ist
- Server gibt durch den **Content-Type-Header** an, was geliefert ist





# Was ist Spring?

- Entwicklung von Enterprise-Anwendungen vereinfachen
- Gute Programmierpraktiken fördern
- Entkopplung der Applikationskomponenten
- Reduzierung von Gluecode und Redundanz



# Was kann Spring?

- Dependency Injection durch IoC Container (Spring Context)
- Integration von Frameworks
  - Keine eigenen Lösungen, wenn es schon gute Frameworks gibt
  - Hibernate, Flyway, JUnit, Mockito, Thymeleaf, Jackson, log4j, ...
- Abstraktionsschichten durch Module
  - Persistierung, Datenzugriff, Sicherheit, MVC, Messaging

# Spring Vorteile

- Entwicklungsumgebung: Spring Tool Suite (STS)
  - Angepasste Eclipse-Umgebung
  - Unterstützt bei der Entwicklung mit Spring Framework und Spring Modulen
- Detaillierte Dokumentation zu allen Modulen
  - Viele Guides
  - Große Community
- Konfiguration
  - XML
  - Annotationen
  - Java-Code
- Module
  - Essenziell für Spring
  - > 200 Repositories auf GitHub
  - Weitere Community Projekte



# Spring Data

- Zugriff auf relationale und NoSQL-Datenbanken
- Basis: JPA
- Zugriff auf Relationale Datenbanken via teilweise selbstimplementierender Repositories (DAOs) => declarative queries
- Reaktiver Zugriff auf NoSQL-Datenbanken JPA-Provider: Hibernate, OpenJPA, EclipseLink, ... DBMS: MySQL, MSSQL, H2, MongoDB, ...



# Spring Security

- Authentifizierung
- AuthenticationManager
  - Erhält Benutzerdaten (Authentication) und entscheidet über Gültigkeit
- UserDetailsService
  - Liefert Benutzerdaten und wird (u. a.) vom AuthenticationManager verwendet
- Autorisierung
  - Stimmt für oder gegen Zugriff auf Ressource



# Spring MVC

- Basiert auf Servlet Engine
- Controller + ViewTemplates
  - Thymeleaf, Velocity, JSP
- REST-Controller dedizierte GUI
  - z. B. Angular
- javax.servlet.http.HttpServlet
  - void doGet(HttpServletRequest request, HttpServletResponse response)  
throws IOException, ServletException
  - void doPost(...)

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

public class HelloSrv extends HttpServlet {
    private int counter = 0;

    // Wird bei HTTP - Get - Anfrage aufgerufen
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
        counter++;

        // Extrahiert Parameter 'name' aus URL
        String name = request.getParameter("name");
        response.setContentType("text/html");

        // Strom für die erzeugte HTML Page
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head><title>Hallo World</title></head>");
        out.println("<body><b>" + counter + ". Hello to " + name +
            "!</b></body>");
        out.println("</html>");

        out.close();
    }
}
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>HelloSrv</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
</web-app>
```

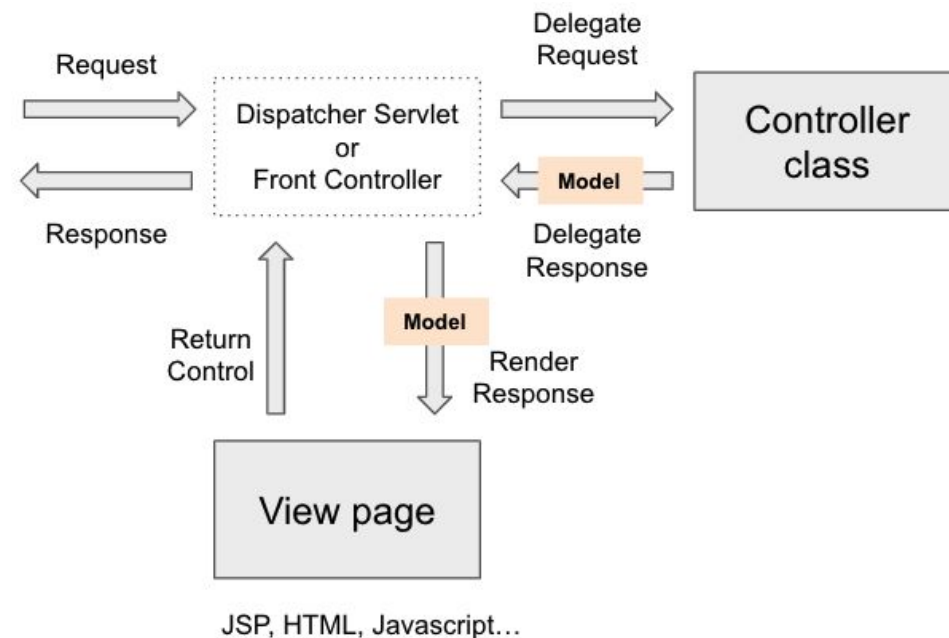
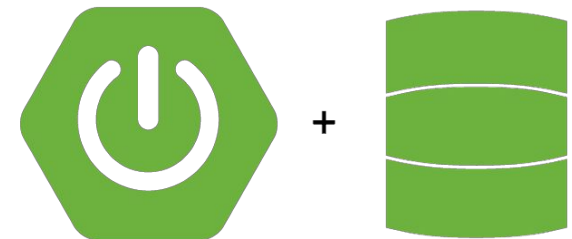


# Spring

- Spring Data Rest Controllers
- @RestController

```
@RestController
public class StudentController {
    private StudentRepository repository;

    @PostMapping("/students")
    Student newStudent(@RequestBody Student student) {
        ...
    }
    ...
}
```



# Welches Problem löst Spring Boot?

- Spring besteht aus sehr vielen Modulen und integriert sehr viele Frameworks
- Es müssen die richtigen Abhängigkeiten in der passenden Version ausgewählt werden
- Jede Komponente muss konfiguriert werden
- Spring Boot wählt kontextabhängig die sinnvollsten Abhängigkeiten und konfiguriert diese automatisch
- Convention over Configuration



# Welches Problem löst Spring Boot?

## Alles konfiguriert und „production ready“

- Runtime: Spring Boot 2.0.3 und Spring Core 5.0.7
- Autorisierung / Authentifizierung: Spring Security 5.0.6
- Container: Tomcat 8.5.31 (embedded) auf Port 80
- Logging: Logback 1.2.3 (Log4J-Nachfolger) via slf4j (Logging-Facade)
- JSON-Binding: Jackson 2.9.6
- Datenhaltung: JPA in einer H2 Datenbank
- View-Engine: Thymeleaf 3.0.9
- YAML-Verarbeitung: SnakeYAML 1.19
- Validierung: Validation API / Hibernate Validator 6.0.10

# Welches Problem löst Spring Boot?

- Spring Boot Starter
  - JAR (per Maven- oder Gradle- Abhängigkeit)
  - Enthält
    - Abhängigkeiten zu benötigten Frameworks
    - Daten zur kontextabhängigen Standardkonfiguration
- Auto Configuration
  - Abhängig von
    - Frameworks im Classpath
    - existierender Konfigurationsdatei



#### Project

☐ Gradle Project ☒ Maven Project

#### Language

☒ Java ☐ Kotlin ☐ Groovy

#### Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (RC2) ☒ 2.7.6 (SNAPSHOT) ☐ 2.7.5  
☐ 2.6.14 (SNAPSHOT) ☐ 2.6.13

#### Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 19 ☒ 17 ☐ 11 ☐ 8

#### Dependencies

ADD DEPENDENCIES... ⌘ + B

#### Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

#### Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.



# Lombok

- Java ist verbose → d.h. man muss viel boilerplate code schreiben
- Lombok = Java Library die generic boilerplate code anhand Annotation generiert

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@ToString(includeFieldNames=true)

```
public class Student {  
    private String name;  
    private String email;  
    private int id;  
}
```