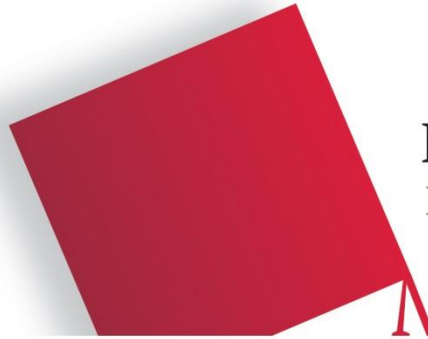


HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

# RESTful Webservices mit JAX-RS

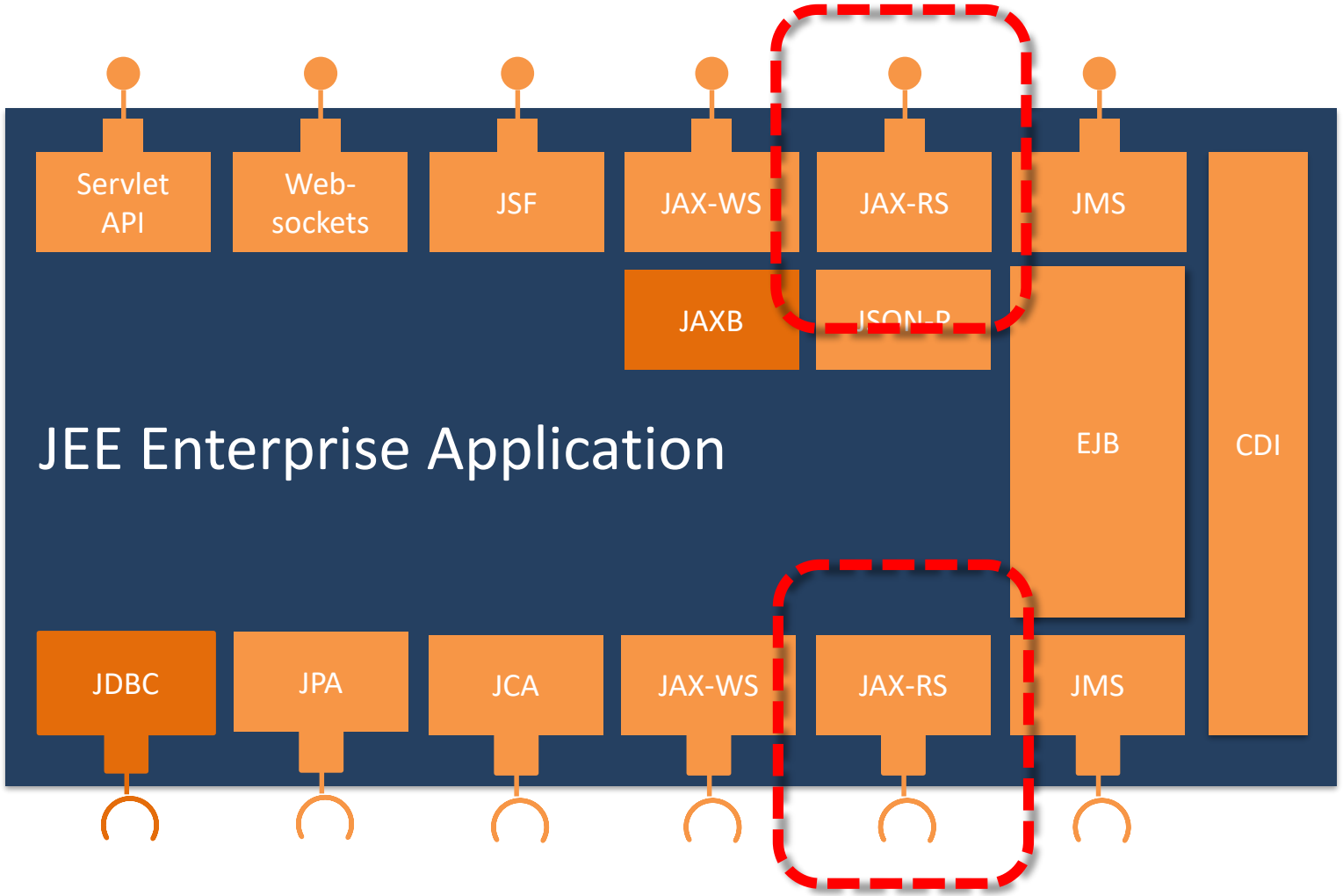
FWP Aktuelle Technologien zur Entwicklung verteilter Java-Anwendungen



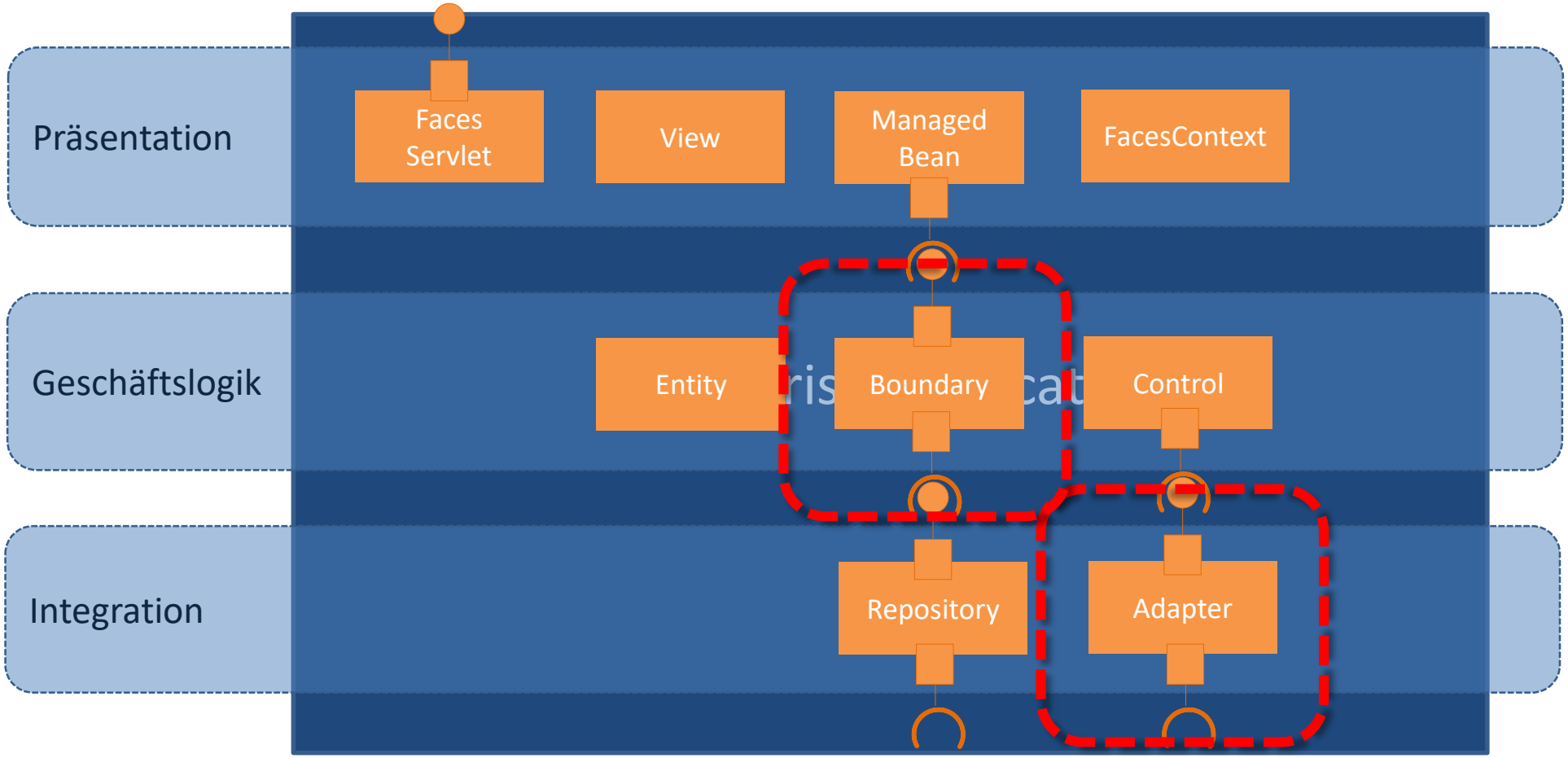
Michael Theis, Lehrbeauftragter

# EINFÜHRUNG IN REST

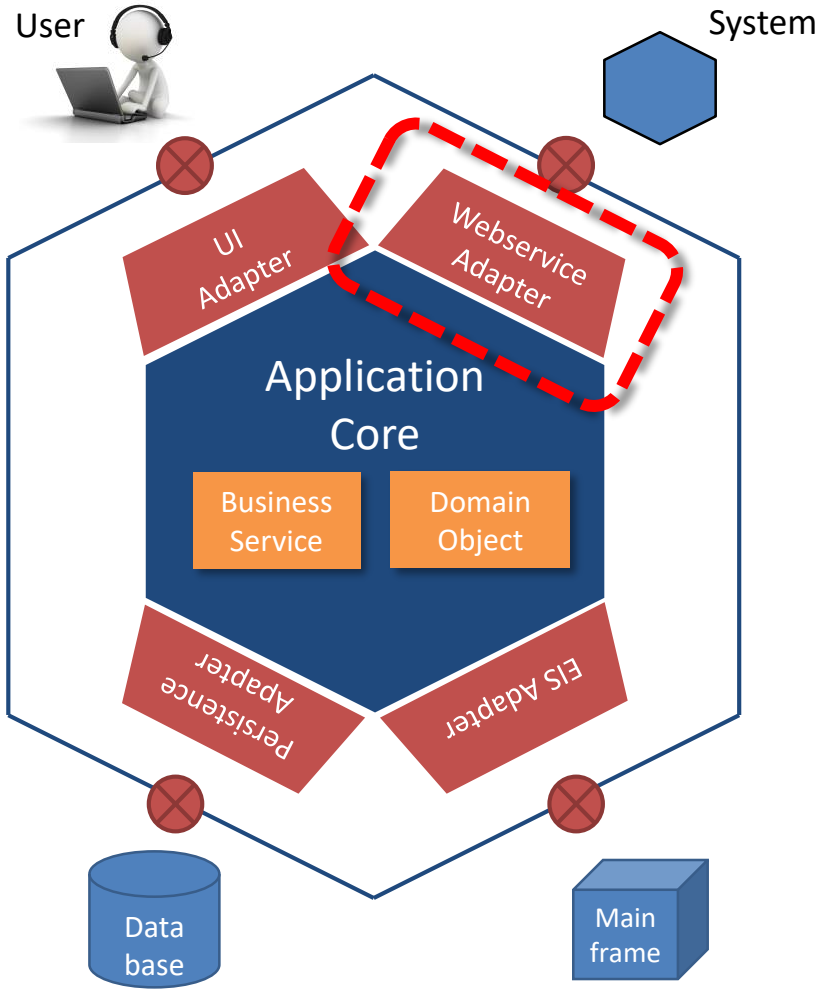
# Kontext: JEE Technologien



# Kontext: Einheitliches Schichtenmodell



# Kontext: Hexagonale Architektur



# Representational State Transfer

- Erstmals beschrieben in Roy Thomas Fieldings Doktorarbeit von 2000:
  **Architectural Styles and the Design of Network-based Software Architectures**
  https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Beschreibt einen Architekturstil für verteilte Systeme basierend auf der Struktur und des Verhaltens des WWWs
- Schwerpunkt liegt auf Maschine-zu-Maschine-Kommunikation

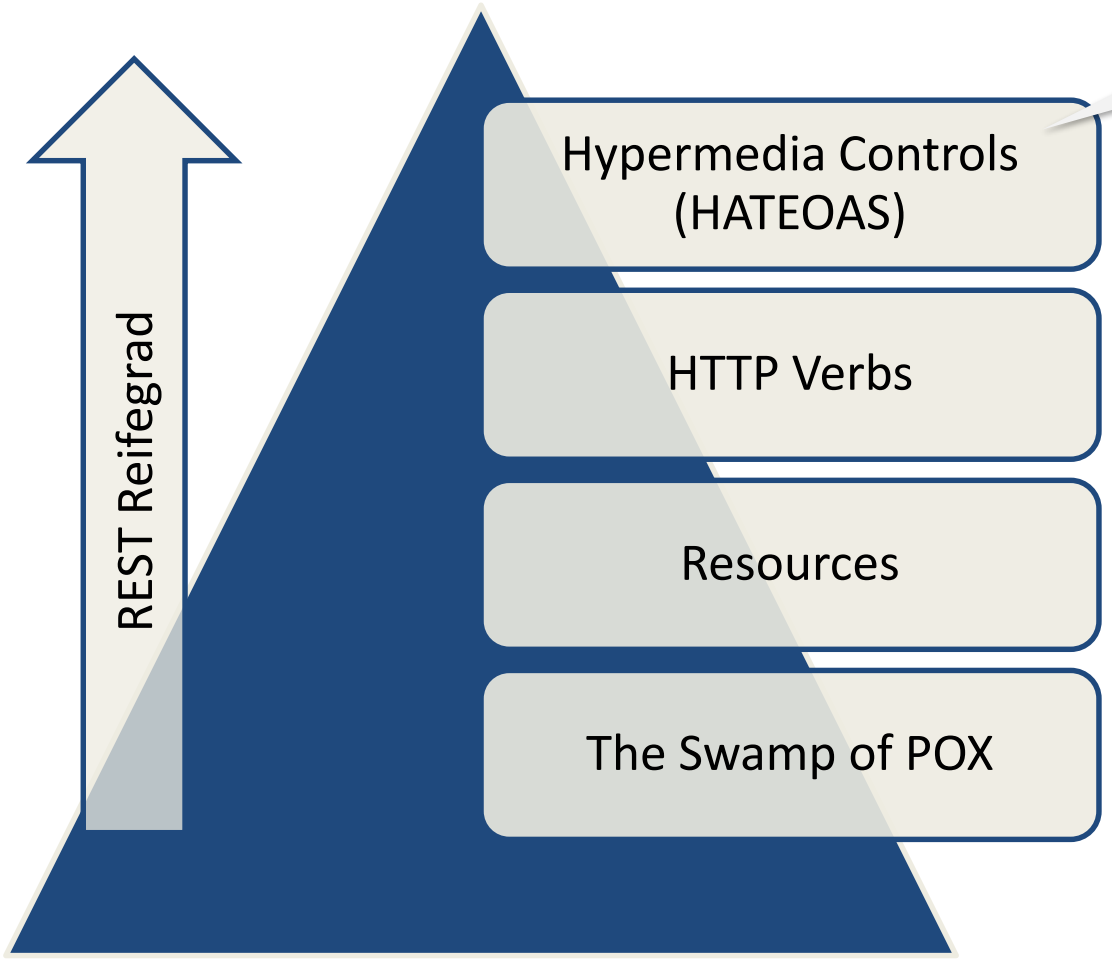
# REST Prinzipien

- Client-Server
- Zustandslosigkeit (Stateless)
- Caching
- Einheitliche Schnittstelle (Uniform Interface)
- Mehrschichtige Systeme (Layered System)
- Code-on-Demand

# Einheitliche Schnittstelle (Uniform Interface)

- Ressource (Resource) als Abstraktion von Information
- Jede Ressource verfügt über einen eindeutigen Bezeichner (Resource Identifier): URI, URL
- Jede Ressource ist über ihren Bezeichner adressierbar
- Jede Ressource kann unterschiedlich dargestellt werden (Representation): JSON, XML, Plain Text,…
- Selbstbeschreibende Nachrichten: URI + HTTP-Verb
- Hypermedia as the Engine of Application State (HATEOAS)

# Richardson Maturity Model




*Wird meist nicht umgesetzt*

- Einsatz von HTTP-Komponenten
- Schnittstelle beschreibt mögliche Operationen
- Antwortnachricht einer Operation ist navigierbar

- HTTP-Verben bestimmen Operation auf der Ressource

- Ressource als Endpunkte für HTTP-Anfragen
- Viele Endpunkte statt einem

- POX: Plain Old XML
- HTTP als Transferprotokoll
- REST als Ersatz für RPC

# HTTP-Verben als Operationen auf Ressourcen

| HTTP-Verb | Ausgelöste Operation |
|---|---|
| GET | Fordert die angegebene Ressource vom Server an.<br>Liefert als Antwort die Repräsentation der Ressourcen zurück. |
| POST | Fügt eine neue Sub-Ressource unterhalb der angegebenen Ressource ein.<br>Liefert als Antwort einen Link auf die hinzugefügte Ressource zurück. |
| PUT | Fügt eine einzelne Ressource neu hinzu (falls sie noch nicht existiert) bzw. aktualisiert die angegebene Ressource (falls sie bereits existiert). |
| PATCH | Ändert einen Teil der angegeben Ressource. |
| DELETE | Löscht die angegebene Ressource. |
| HEAD | Liefert die Metadaten zu der angegebenen Ressource zurück. |
| OPTIONS | Liefert eine Liste von Operationen zurück, die für die angegebene Ressource angeboten werden. |
| TRACE | Liefert die Anfrage so zurück, wie sie beim Server angekommen ist. |

# Content Negotiation über MIME-Typen

- Client und Server handeln Datenformat des Payloads aus:
  - HTTP request header Accept mit gewünschten Datenformaten
  - HTTP response header Content-Type mit tatsächlichem Datenformat
- Resource im Server bestimmt mit @Produces, welche Datenformate bei Responses unterstützt werden
- Resource im Server bestimmt mit @Consumes, welche Datenformate bei Requests unterstützt werden
- Bester Match gewinnt, sonst HTTP status code 406 Not Acceptable

Wie exportiere ich RESTful Webservices in einer JEE-Applikation mit JAX-RS?

# BEREITSTELLEN VON REST SERVICES

# Einschalten von JAX-RS

- JAX-RS wird mit jedem JEE Application Server ausgeliefert, muss aber explizit eingeschaltet werden
- Eigene Unterklasse von javax.ws.rs.core.Application erstellen

```
@ApplicationPath("rest")
public class RestApplicationConfig extends Application {

}
```

  - JAX-RS erkennt nun annotierte REST Ressource-Klassen
  - @ApplicationPath bestimmt den gemeinsamen Pfad zu den REST Ressourcen

# @Path bestimmt Ressource und deren Pfad

- Ressource = POJO-Klasse + @Path

```
@Path(„tasks")
public class TasksResource {
```

- Ressource = Scoped CDI-Bean-Klasse + @Path

```
@Path(„tasks")
@RequestScoped // oder @Singleton sonst funktioniert DI nicht !!!
public class TasksResource {
```

- Ressource = Stateless Session Bean-Klasse + @Path

```
@Path(„tasks")
@Stateless
public class TasksResource {
```

- Ressourcenname sollte immer Substantiv im Plural sein!

# Mapping von Requests auf Methoden

- @Path auf Klassenebene bestimmt Pfad der Ressource
- @GET, @POST, @PUT, @DELETE bestimmen, welche Methode welchen Request entgegennimmt
- @Path auf Methodenebene definiert Unterpfade zu @Path auf Klassenebene

```
@Path("tasks")
public class TasksResource {
    @GET
    @Path("{taskId}") // Methode hört auf */tasks/9999
    public Task getTaskById(@PathParam("taskId") long taskId) {
        …
```

# Mapping von Request-Parametern

- @PathParam mappt Parameter aus dem Request-Pfad

```
GET http://localhost:8080/jeedemo/rest/tasks/1234
@Path("tasks")
public class TasksResource {
    @GET @Path("{taskId}")
    public Task getTaskById(@PathParam("taskId") long taskId) {
        …
```

- @QueryParam mappt Parameter aus dem Query-Teil

```
POST http://localhost:8080/jeedemo/rest/tasks?returnEntity=true
@Path("tasks")
public class TasksResource {
    @POST
    public Response createTask(Task newTask,
                               @QueryParam(„returnEntity") boolean returnEntity) {
        …
```

# Mapping von Rückgabewerten

- Implizites Mapping über direkten Rückgabewert

```
@GET
@Path("{taskId}")
public Task getTaskById(@PathParam("taskId") long taskId) {
    return this.repository.findTaskById(taskId);
}
```

Und was passiert bei einem Fehler?

- Explizites Mapping mit javax.ws.rs.core.Response **(empfohlen)**

```
@GET
@Path("{taskId}")
public Response getTaskById(@PathParam("taskId") long taskId) {
    return Response.ok(this.repository.findTaskById(taskId)).build();
}
```

Fehlerbehandlung mit sinnvollen HTTP Status Codes möglich

# Anforderungen an Entitäten (= Payload)

- Marshalling/Unmarshalling erfolgt über ContentHandler
- Meist verwendeten Formate JSON und XML werden automatisch unterstützt
- Binding der Java-Objekte an JSON oder XML erfolgt über JAX-B
- Java-Klassen müssen die folgenden Anforderungen erfüllen
  - Default-Konstruktor (zwingend erforderlich für JAX-B !!!)
  - JAX-B-Annotationen auf Klassenebene: @XmlRootElement, @XmlAccessorType

Wie rufe ich RESTful Webservices mit JAX-RS auf?

# KONSUMIEREN VON REST SERVICES

# REST Client API im Schnelldurchgang

- javax.ws.rs.client.ClientBuilder#newClient() erstellt Clients
- javax.ws.rs.client.Client#target() erstellt neue Endpunkte (WebTarget) für REST-Requests
- javax.ws.rs.client.WebTarget#path() setzt Pfad zur Ressource
- javax.ws.rs.client.WebTarget#request() erstellt Request-Builder
- Builder#get()|put()|post()|delete() setzt HTTP-Request ab
- Kontrolle über Header, Parameter, Cookies, Payloads möglich
- Asynchrone Aufrufen werden auch unterstützt

# Fluent API macht REST Aufrufe einfach

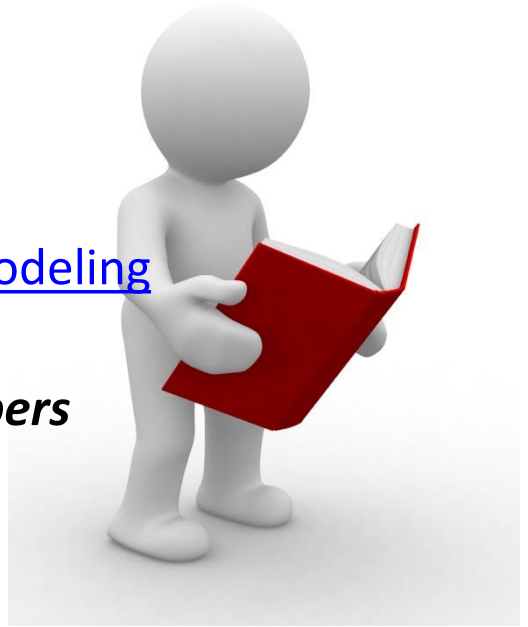
```
String responsePayload =
    ClientBuilder \\
        .newClient() \\ erzeuge neuen Client
        .target("http://localhost:8080/jeedemo/rest/hello") \\ setze Ressource-URL
        .request(MediaType.TEXT_HTML) \\ erzeuge Request mit HTML als Datenformat
        .get(String.class); \\ führe GET Request mit Unmarshalling des Payloads aus
```

# Fragen?

# ANHANG

# Quellen

- Beispiel-Code auf GitHub unter https://github.com/mikeT92/jeetrain
  MAVEN-Projekt **jeedemo** MAVEN-Modul **jeedemo-jaxrs**
- ***The Java EE Tutorial Part VI Webservices Chapter 32 ff.***
  https://javaee.github.io/tutorial/jaxrs.html#GIEPU
- Stefan Tilkov: ***REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web***
  dpunkt.verlag GmbH; 3. Auflage (30. April 2015); ISBN 978-3864901201
- ThoughtWorks Prakash Subramaniam: ***REST API Design – Resource Modeling***
  https://www.thoughtworks.com/de/insights/blog/rest-api-design-resource-modeling
  abgerufen am 11.02.2018
- Stormpath Les Hazelwood: ***REST+JSON API Design - Best Practices for Developers***
  https://youtu.be/hdSrT4yjS1g
  abgerufen am 11.02.2018



# Kontakt

**Michael Theis**
Lehrbeauftragter Hochschule München

email michael.theis@hm.edu
mobile + 49 170 5403805
web http://www.tschutschu.de/Lehrauftrag.html