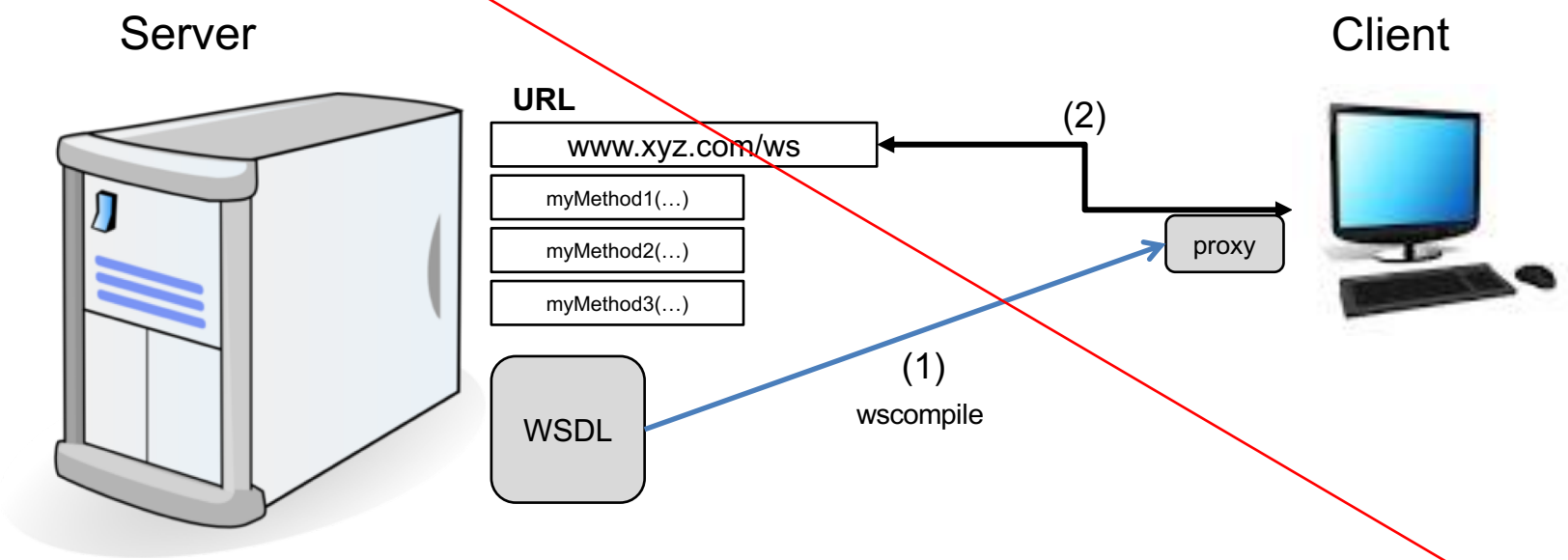
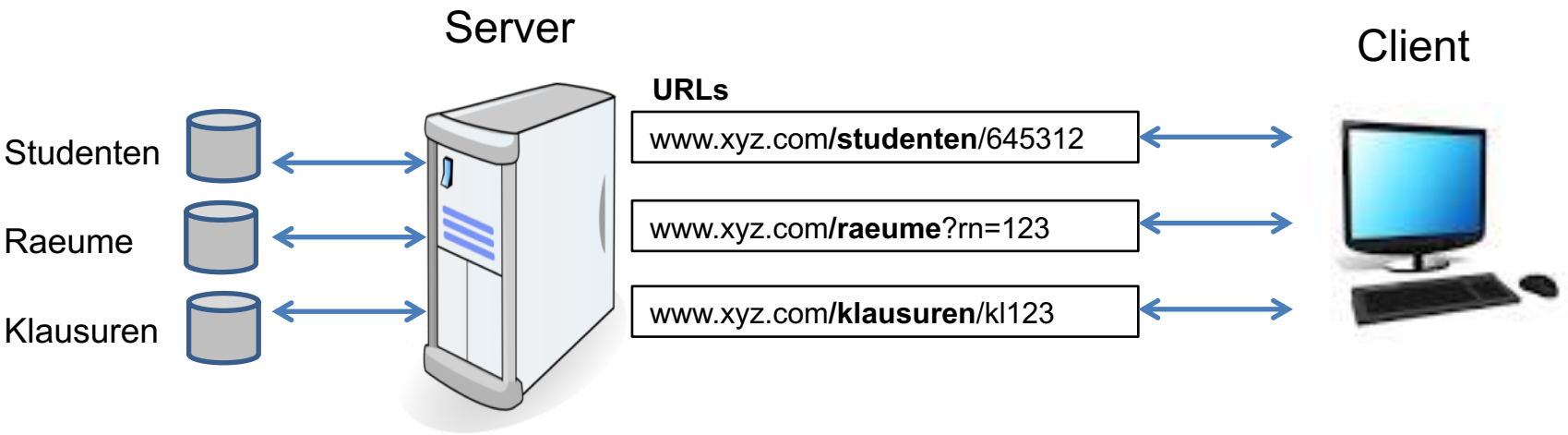
Web-Services

Modul 14
Web-Services: SOAP

SOAP (*Simple Object Access Protocol)*



REST (Representational State Transfer)



REST (Representational State Transfer)

Request

```
http://www.acme.com/phonebook/UserDetails/12345
```

```
http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe
```

Response

```
{
  "id": "12345"
  "name":"Hans",
  "vorname":"Mueller",
  "phone":"0123 456789"
}
```

JSON

or

```
<parts-list>
 <part id="3322">
  <name>ACME Boomerang</name>
  <desc>
   Used by Coyote in <i>Zoom at the Top</i>, 1962
  </desc>
  <price currency="usd" quantity="1">17.32</price>
  <uri>http://www.acme.com/parts/3322</uri>
 </part>
 <part id="783">
  <name>ACME Dehydrated Boulders</name>
  <desc>
   Used by Coyote in <i>Scrambled Aches</i>, 1957
  </desc>
  <price currency="usd" quantity="pack">19.95</price>
  <uri>http://www.acme.com/parts/783</uri>
 </part>
</parts-list>
```

XML

or …

Modul 14
REST-Architektur

Adressierbarkeit: Jede Ressource muss über einen eindeutigen ***Unique Resource Identifier*** (kurz URI) identifiziert werden können. Ein Kunde mit der Kundennummer `123456` könnte also zum Beispiel über die URI `http://ws.mydomain.tld/customers/123456` adressiert werden.

Zustandslosigkeit: Die Kommunikation der Teilnehmer untereinander ist zustandslos. Dies bedeutet, dass **keine Benutzersitzungen** (etwa in Form von Sessions und Cookies) existieren, sondern bei jeder Anfrage alle notwendigen Informationen wieder neu mitgeschickt werden müssen..

Einheitliche Schnittstelle: Jede Ressource muss über einen einheitlichen Satz von Standardmethoden zugegriffen werden können. Beispiele für solche Methoden sind die Standard-HTTP-Methoden wie `GET`, `POST`, `PUT`, und mehr.

Entkopplung von Ressourcen und Repräsentation: Das bedeutet, dass verschiedene Repräsentationen einer Ressource existieren können. Ein Client kann somit etwa eine Ressource explizit beispielsweise im XML- oder JSON-Format anfordern.

Quelle: https://blog.mittwald.de/webentwicklung/restful-webservices-1-was-ist-das-uberhaupt/

Modul 14
HTTP-Request-Methoden

URL

```
http://xyz.de/Artikel/Buecher/4711
```

HTTP	CRUD	Beispiel-URL und -Bedeutung
GET	Read	**`http://xyz.de/Artikel/Buecher`** --> Liste aller Bücher; **`http://xyz.de/Artikel/Buecher/4711`** --> Informationen zu dem per ID ausgewählten Buch; **`http://xyz.de/Artikel/Buecher?isbn=1234567890`** --> Informationen zu dem per Suchkriterium ausgewählten Buch
POST	Create	**`http://xyz.de/Artikel/Buecher`** --> Neuen Artikel hinzufügen (mit neuer ID) (dabei wird üblicherweise die automatisch vergebene ID returniert)
PUT	Update, Create	**`http://xyz.de/Artikel/Buecher/4711`** --> Update (oder Create) des per ID identifizierten Artikels
DELETE	Delete	**`http://xyz.de/Artikel/Buecher/4711`** --> Diesen per ID identifizierten Artikel löschen
…	…	…

Beispielabfrage: https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=Music

Modul 14
Safe and / or idempotent methods

4.2.1. Safe Methods

Request methods are considered "safe" if their defined semantics are essentially read-only; i.e., the client does not request, and does not expect, any state change on the origin server as a result of applying a safe method to a target resource. [...]
Of the request methods defined by this specification,
the GET, HEAD, OPTIONS, and TRACE methods are defined to be safe. [...]

4.2.2. Idempotent Methods

A request method is considered "idempotent" if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request. Of the request methods defined by this specification, PUT, DELETE, and safe request methods are idempotent. [...]

Safe methods are automatically idempotent

Modul 14
Safe and / or idempotent methods

4.2.1. Safe Methods

READ-ONLY

GET, HEAD, OPTIONS TRACE

4.2.2. Idempotent Methods

Serverseitig:
Mehrmaliger Aufruf == Einmaliger Aufruf

PUT, DELETE

Modul 14
Safe and / or idempotent methods

Per Konvention soll (sollte) gelten:

Method	Safe	Idempotent	
CONNECT	no	no	
DELETE	no	yes	Löschen
GET	yes	yes	Lesen
HEAD	yes	yes	
OPTIONS	yes	yes	
POST	no	no	Create
PUT	no	yes	Update
TRACE	yes	yes	

Modul 14
Safe and / or idempotent methods

Man liest oft:

An idempotent HTTP method is a HTTP method that can be called many times without different outcomes.

(Quelle: http://restcookbook.com/HTTP%20Methods/idempotency/)

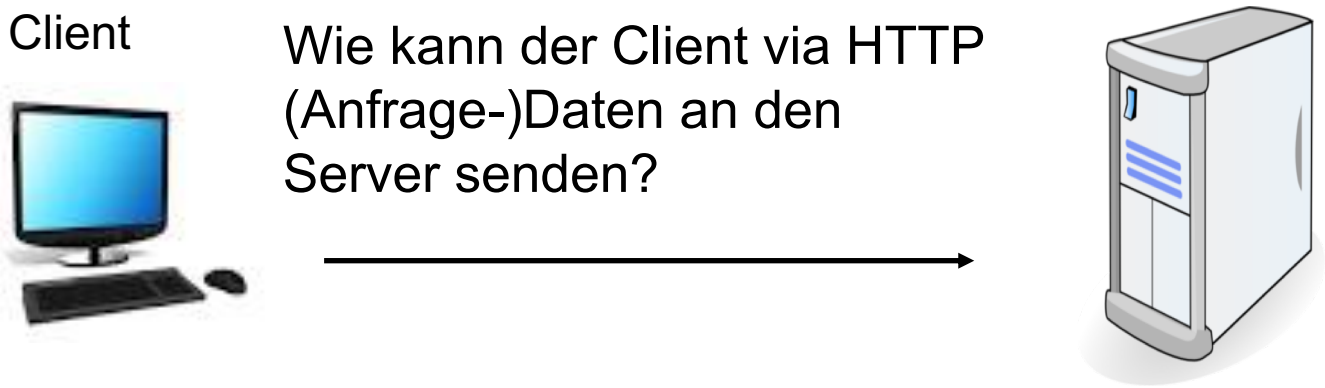
Unsinn!
Abfrage der aktuellen Uhrzeit liefert immer einen anderen Output.
Aber: Serverseitig gleicher Zustand nach einem oder mehreren Aufrufen. Weiterhin reine Lese-Operation. Somit via `HTTP-GET` zulässig.

Server Side

(RSET-Endpoint)

```
http://www.xyz.de/halloworld
```

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```



Vier Möglichkeiten:

1. Über die URL (Path):
   http://localhost:80/proj/Mueller
   http://localhost:80/proj/Meier
2. Über Query-Parameter
   http://localhost:80/proj?name=Mueller
3. Über Formular-Daten
4. Über Daten im Request-Body

Modul 14
How to receive data from the client

HTTP-Method	Options to receive data
GET	via PathParameter: @PathParam via QueryParameter: @QueryParam
DELETE	via PathParameter: @PathParam via QueryParameter: @QueryParam
POST	via PathParameter: @PathParam via QueryParameter: @QueryParam via Formular-Data: @FormData via Request-Body: "..."
PUT	via PathParameter: @PathParam via QueryParameter: @QueryParam via Formular-Data: @FormData via Request-Body: "..."

Quelle: http://docs.jboss.org/resteasy/docs/1.1.GA/userguide/html/Using__Path_and__GET___POST__etc..html

Modul 14
The `@PathParam` Annotation and URI Path Templates

Beispiel: http://www.xyz.de/users/*grabowski*

```
@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("username") String userName) {
        ...
    }
}
```

```
@Path("/{name1}/{name2}/")
public class SomeResource {
    ...
}
```

URI Template	URI After Substitution
`http://example.com/{name1}/{name2}`	`http://example.com/james/gatz`
`http://example.com/{z1}/{z2}/{z3}`	`http://example.com/7/55/13`

Quelle: http://docs.oracle.com/javaee/6/tutorial/doc/gilik.html

Modul 14
The `@QueryParam` Annotation

Beispiel: http://www.xyz.de/users?id=*grabowski*

```
@Path("/users")
public class UserResource {
    @GET
    @Produces("text/xml")
    public String getUser(@QueryParam("id") String userName) {
        ...
    }
}
```

Beispiel: http://www.xyz.de/users?id=*mueller&iq=220*

```
@Path("/users")
public class UserResource {
    @GET
    @Produces("text/xml")
    public String getUser(@QueryParam("id") String userName,
                          @QueryParam("iq") String smart) {
        ...
    }
}
```

Quelle: http://docs.oracle.com/javaee/6/tutorial/doc/gilik.html

Modul 14

`@GET` and `@DELETE` – Transfer data by URI

```
@Path("/library")
public class Library {

   @GET
   @Path("/books")
   public String getBooks() {...}

   @GET
   @Path("/book/{isbn}")
   public String getBook(@PathParam("isbn") String id,
                         @QueryParam("iq") String smart ) {
      // Get data from the Path-Parameter – access by String id
   }
}
```

Root path: **`http://myhost.com/services`**

`GET http://myhost.com/services/library/books`
`GET http://myhost.com/services/library/book/333?iq=222`

Quelle: http://docs.jboss.org/resteasy/docs/1.1.GA/userguide/html/Using__Path_and__GET___POST__etc..html

Modul 14

`@POST` and `@PUT` – Transfer data by URI

```
@Path("/library")
public class Library {

   @POST
   @Path("/books")
   public String getBooks() {...}

   @POST
   @Path("/book/{isbn}")
   public String getBook(@PathParam("isbn") String id,
                         @QueryParam("iq") String smart ) {
      // Get data from the Path-Parameter – access by String id
   }
}
```

Root path: **`http://myhost.com/services`**

`GET http://myhost.com/services/library/books`
`GET http://myhost.com/services/library/book/333?iq=222`

Quelle: http://docs.jboss.org/resteasy/docs/1.1.GA/userguide/html/Using__Path_and__GET___POST__etc..html

Modul 14

@POST and @PUT - Transfer data with form-data

```
@Path("/user/add")
public class UserService {
        @POST
        @Produces("text/html")
        @Consumes("application/x-www-form-urlencoded”)
        public String addUser(
                @FormParam("name") String name,
                @FormParam("age") int age) {
                …
                return "ok";
        }
}
```

```
<html>
<body>
        <h1>JAX-RS @FormQuery Testing</h1>
        <form action="rest/user/add" method="post">
                <p> Name : <input type="text" name="name" />
                </p>
                <p>Age : <input type="text" name="age" />
                </p>
                <input type="submit" value="Add User" />
        </form>
 </body>
</html>
```

Quelle: http://docs.jboss.org/resteasy/docs/1.1.GA/userguide/html/Using__Path_and__GET___POST__etc..html

Modul 14

@POST and @PUT - Transfer data by Request-Body

```
@Path("/library")
public class Library {

   @POST
   @Path("/books")
   public String getBooks(String thePostedData) {
      // Data is send via request body - access from String thePostedData
      System.out.println("Data: ", thePostedData);
   }

   @POST
   @Path("/book/{isbn}")
   public String getBook(@PathParam("isbn") String id, String postedData) {
       // Data is send via request body - access by String thePostedData
       // Get data from the Path-Parameter - access by String id
   }
}
```

The POST-Request can not be executed dircetly with the browser. An additional tool is required (e.g.Insomnia).

Quelle: http://docs.jboss.org/resteasy/docs/1.1.GA/userguide/html/Using__Path_and__GET___POST__etc..html

Modul 14
`@Produces`

```
@Path("/myResource")
@Produces("text/plain")
public class SomeResource {
    @GET
    public String doGetAsPlainText() {
        ...
    }

    @GET
    @Produces("text/html")
    public String doGetAsHtml() {
        ...
    }
}
```

```
@Produces({"image/jpeg, image/png"})
```

```
@Produces({"application/xml", "application/json"})
```

The @Produces Annotation just adds a content tag to the header. No check if the data really matches to the exposed format.

Quelle: http://docs.oracle.com/javaee/6/tutorial/doc/gilik.html

Modul 14
`@Consumes`

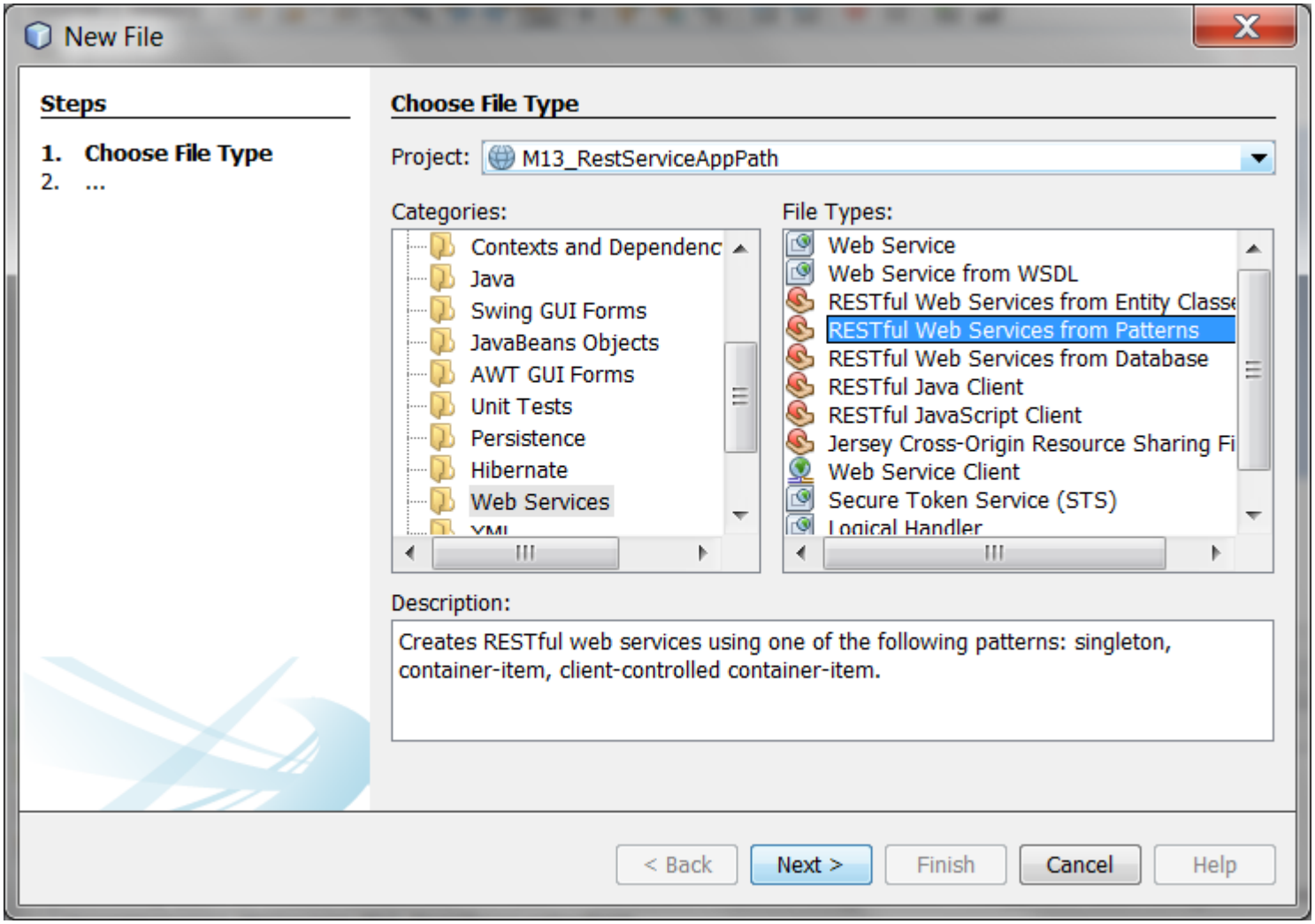
```
@Path("/myResource")
@Consumes("text/plain")
public class SomeResource {
    @POST
    public String doPost(String msg) {
        ...
    }

    @POST
    @Consumes("text/html")
    public String doPost2(String htmlMsg) {
        ...
    }
}
```

```
@Consumes({"text/plain, text/html"})
```

The @Consumes Annotation only makes sense in combination with POST / PUT, when data is transported in the body. However, no check whether the data really matches the demanded format.

Quelle: http://docs.oracle.com/javaee/6/tutorial/doc/gilik.html

Modul 14
JAX-RS Example1



Quelle: http://docs.oracle.com/javaee/6/tutorial/doc/gilik.html

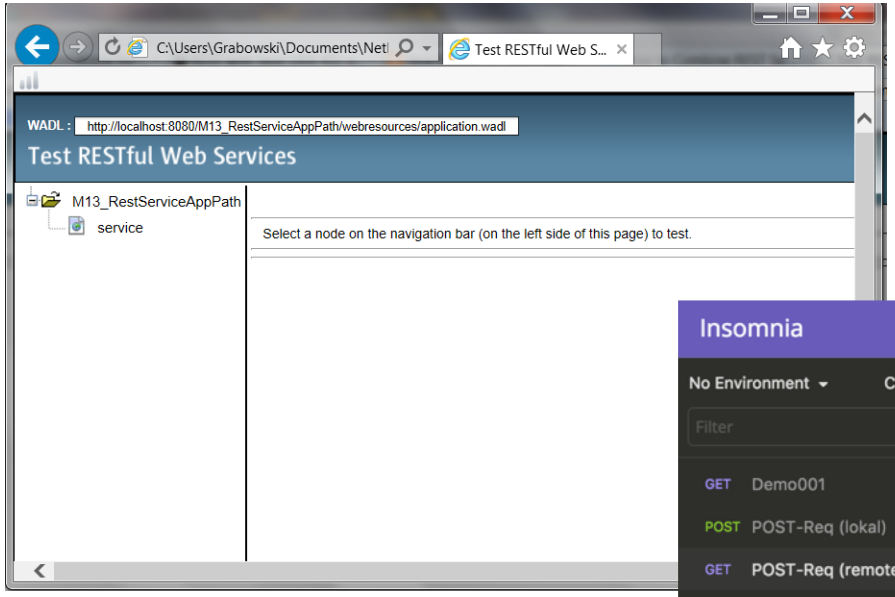
Modul 14
JAX-RS Example1

```
@Path("service")
public class ServiceResource {

    public ServiceResource() {
    }

    @GET
    @Produces("text/html")
    public String getText() {
        return "Hallo my friend!";
    }
}
```

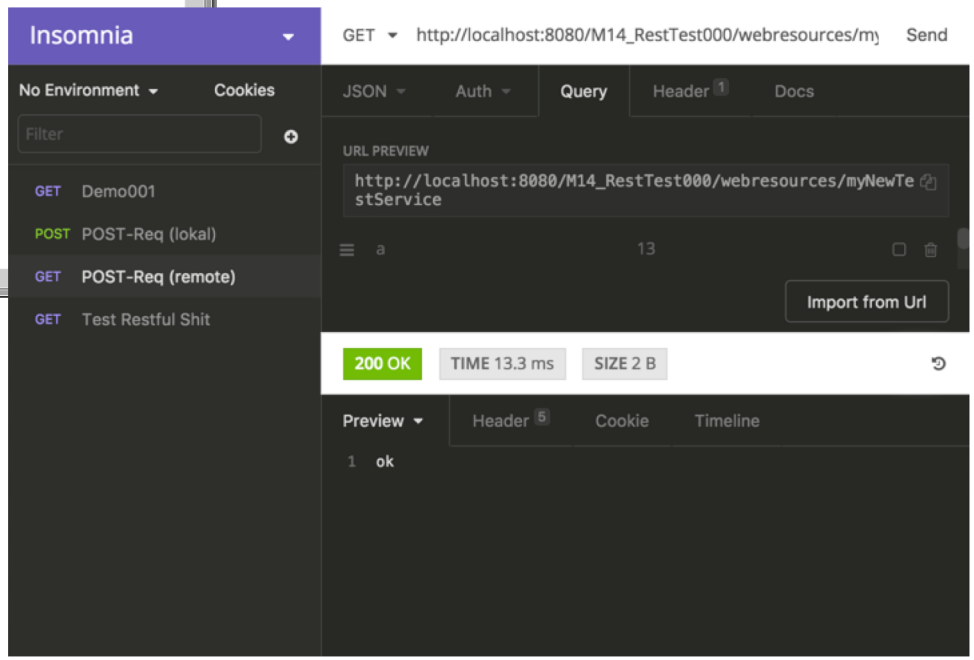
Quelle: http://docs.oracle.com/javaee/6/tutorial/doc/gilik.html

Modul 14
JAX-RS Example1

Test RESTful WebServices



Part of netbeans



Third party tool

Quelle: http://docs.oracle.com/javaee/6/tutorial/doc/gilik.html

Client Side

Bsp.: https://en.wikipedia.org/w/api.php?action=query&prop=extracts&rvprop=content&rvsection=0&titles=pizza

Options for building a REST Client

Options

- **Commons HTTP Client** build your own
- **Apache CXF** has three different REST Client options
- **Spring** also has its own called RestTemplate
- **Ning Async-http-client.**
- **Jersey** (JAX-RS Referenzimplementierung)

Modul 14
Building a REST Client with Jersey 2.x

```
public static void main(String[] args) {

    Client client = ClientBuilder.newClient();

    WebTarget webTarget = client.target("https://en.wikipedia.org/w/api.php");

    WebTarget resourceWebTarget = webTarget.queryParam("action", "query")
                                           .queryParam("prop", "extracts")
                                           .queryParam("format", "json")
                                           .queryParam("titles", "Music");


    Invocation.Builder invocationBuilder =
    resourceWebTarget.request(MediaType.APPLICATION_JSON);

    Response response = invocationBuilder.get();

    String erg = response.readEntity(String.class);

    System.out.println(erg);

}
```

Example:
Wikipedia Request

Modul 14
Jersey 2.x RESTClient: HTTP-GET / HTTP-DELETE

```
Client client = ClientBuilder.newClient();

// Read or Delete of data
// Aufruf von https://www.xyz.de/hans/mueller?id=123 via HTTP-GET oder HTTP-DELETE
WebTarget webTarget = client.target("https://www.xyz.de");

WebTarget resourceWebTarget = webTarget.path("hans")
                                       .path("mueller")
                                       .queryParam("id", "123");

Invocation.Builder invocationBuilder =
resourceWebTarget.request(MediaType.APPLICATION_JSON);

Response response = invocationBuilder.get();
// Via delete: response = invocationBuilder.delete();

String erg = response.readEntity(String.class);

System.out.println(erg);
```

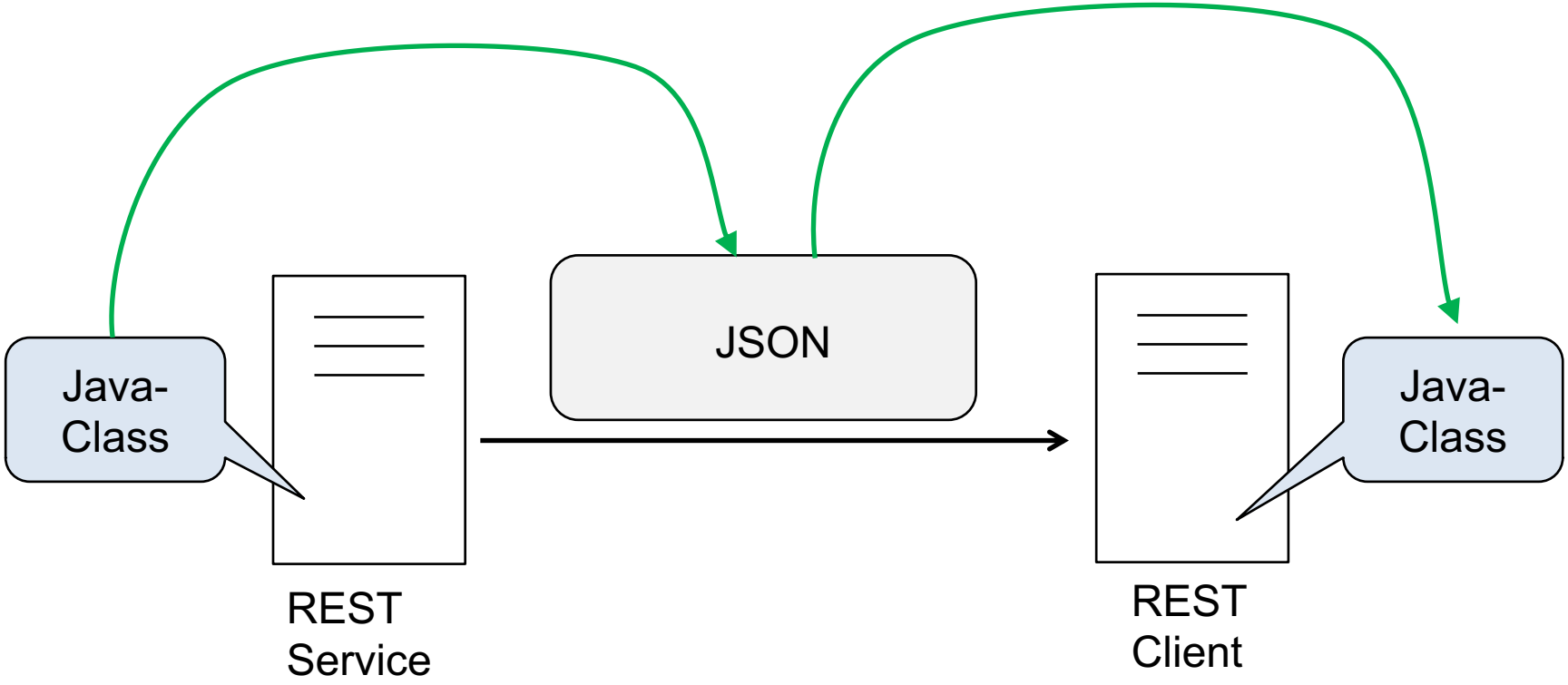
Modul 14
Jersey 2.x RESTClient: HTTP-POST / HTTP-PUT

```
// Submit data in body directly
 public static void restViaPost() {

    Client client = ClientBuilder.newClient();

    // Create or Update of data
    // Aufruf von https://www.xyz.de/hans/mueller via HTTP-POST oder HTTP-PUT
    WebTarget webTarget = client.target("https://www.xyz.de");

    WebTarget resourceWebTarget = webTarget.path("hans")
                                           .path("mueller");

    Invocation.Builder invocationBuilder =
          resourceWebTarget.request(MediaType.APPLICATION_JSON);

    Response response = invocationBuilder
                        .post(Entity.entity("{ \"id\": \"123\" }",
                                            MediaType.APPLICATION_JSON));

    String erg = response.readEntity(String.class);

    System.out.println(erg);

}
```

Modul 14
Jersey 2.x RESTClient: HTTP-POST / HTTP-PUT

```
// Submit data via formParametes
public static void restViaPost2() {

    Client client = ClientBuilder.newClient();

    // Create or Update of data
    // Aufruf von https://www.xyz.de/hans/mueller via HTTP-POST oder HTTP-PUT
    WebTarget webTarget = client.target("https://www.xyz.de");

    WebTarget resourceWebTarget = webTarget.path("hans")
                                           .path("mueller");

    Invocation.Builder invocationBuilder =
                    resourceWebTarget.request(MediaType.APPLICATION_JSON);

    Form form = new Form();
    form.param("id", "123");
    Response response = invocationBuilder
                        .post(Entity.entity(form, MediaType.APPLICATION_JSON));

    String erg = response.readEntity(String.class);

    System.out.println(erg);

}
```

Modul 14
Converting POJO to / from JSON



Verwendung einer Library – z.B. JACKSON

Quelle: http://entjavastuff.blogspot.de/2012/07/jax-rs-client-beyond-hello-world.html

Modul 14
JACKSON - Get value from a certain key

```
{
    "batchcomplete": "",
    "query": {
        "pages": {
            "8983183": {
                "pageid": 8983183,
                "ns": 0,
                "title": "Money",
                "extract": "<p><b>Money<\/b> is any item
            }
        }
    }
}
```

jsonStr

Get value from the key „extract“

```
JsonNode root = mapper.readTree(jsonStr);

String wikiTxt = root.findValue("extract").asText();
```

Modul 14
JACKSON – Convert Java object to JSON

```
ObjectMapper mapper = new ObjectMapper();
Staff obj = new Staff();

//Object to JSON in file
mapper.writeValue(new File("c:\\file.json"), obj);

//Object to JSON in String
String jsonInString = mapper.writeValueAsString(obj);
```


Source: https://www.mkyong.com/java/jackson-2-convert-java-object-to-from-json/

Modul 14
JACKSON – Convert JSON to Java object

```
ObjectMapper mapper = new ObjectMapper();
String jsonInString = "{'name' : 'mkyong'}";

//JSON from file to Object
Staff obj = mapper.readValue(new File("c:\\file.json"), Staff.class);

//JSON from URL to Object
Staff obj = mapper.readValue(new URL("http://mkyong.com/api/staff.json"), Staff.class);

//JSON from String to Object
Staff obj = mapper.readValue(jsonInString, Staff.class);
```


Source: https://www.mkyong.com/java/jackson-2-convert-java-object-to-from-json/

Modul 14
Java Object from large JSON files

Remark: How to generate the Java Object from JSON?


```
"kind": "books#volumes",
"totalItems": 679,
"items": [
 {
  "kind": "books#volume",
  "id": "7ht_t5GOI7kC",
  "etag": "qRSudfbQ99Q",
  "selfLink": "https://www.googleapis.com/books/v1/volumes/7ht_t5GOI7kC",
  "volumeInfo": {
   "title": "Festschrift für Robert Fischer",
   "authors": [
    "Robert Fischer",
    "Marcus Lutter",
    "Walter Stimpel",
    "Herbert Wiedemann"
   ],
   "publisher": "Walter de Gruyter",
   "publishedDate": "1979",
   "industryIdentifiers": [
    {
     "type": "ISBN_10",
     "identifier": "3110074788"
    },
    {
     "type": "ISBN_13",
     "identifier": "9783110074789"
    }
   ],
   "pageCount": 928,
   "printType": "BOOK",
   "categories": [
    "Law"
```


Help @: http://www.jsonschema2pojo.org/

Quelle: http://entjavastuff.blogspot.de/2012/07/jax-rs-client-beyond-hello-world.html