

RESTful Web

Representational State Transfer

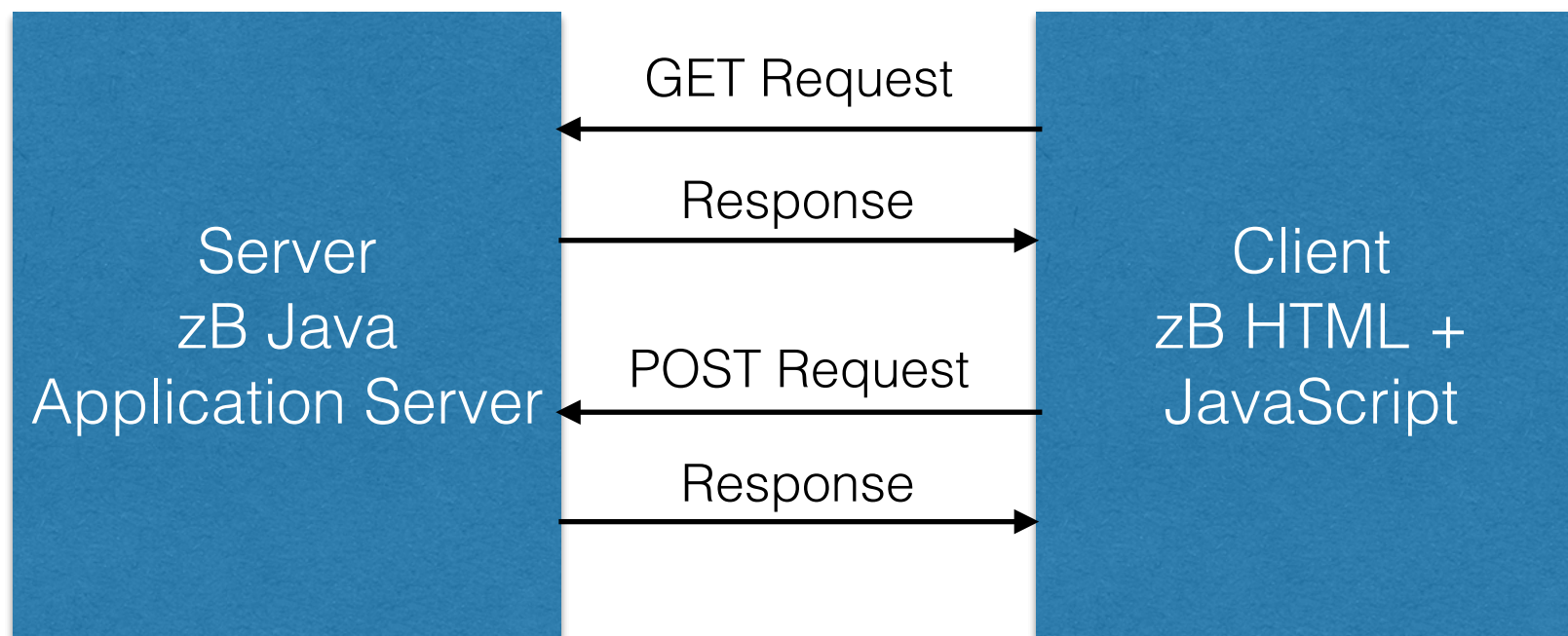
Warum REST?

- REST ist die Lingua Franca des Webs
- Heterogene (verschiedenartige) Systeme können mit REST kommunizieren, unabhängig von Technologie der beteiligten Systeme
- REST Ressourcen werden im Web angeboten: zB Google Maps, Twitter, Youtube, Facebook

<http://www.programmableweb.com/apis/directory>

Funktionsweise

- Sämtliche Informationen im Web sind Ressourcen, auf die mittels HTTP zugegriffen werden kann.



Anforderungen

- **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://localhost/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 dass bei jeder Anfrage alle notwendigen Informationen wieder neu mitgeschickt werden müssen.
- **Einheitliche Schnittstelle:** Auf 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.

HTTP Methoden

GET

READ: Lesen einer bestimmten Ressource

POST

CREATE (und UPDATE): Erstellen oder Ändern
ohne einer ID

PUT

(CREATE und) UPDATE: Erstellen oder Ändern
mit einer bekannten ID

DELETE

DELETE: Löschen einer Ressource

Beispiele













Request	Response
<pre>GET /products HTTP/1.0 Accept: application/json</pre>	<pre>HTTP/1.0 200 OK Content-Type: application/json Content-Length: 1234 [{ uri: "http://ws.mydomain.tld/products/1000", name: "Gartenstuhl", price: 24.99 }, { uri: "http://ws.mydomain.tld/products/1001", name: "Sonnenschirm", price: 49.99 }]</pre>

Beispiel POST

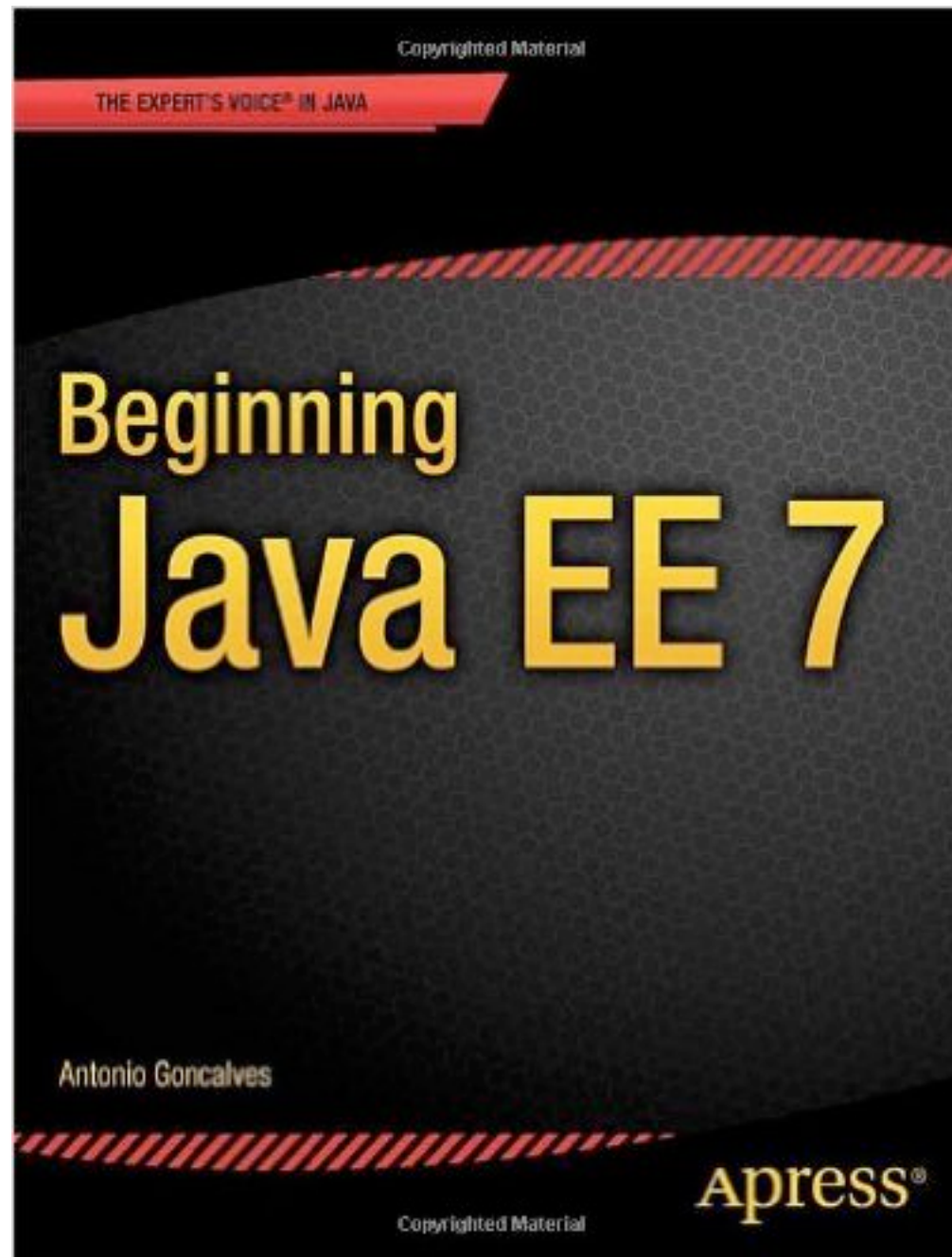
Request	Response
<pre>POST /products HTTP/1.0 Content-Type: application/json Content-Length: 38 { name: "Sandkasten", price: 89.99 }</pre>	<pre>HTTP/1.0 201 Created Location: http://ws.mydomain.tld/products/1002</pre>

Begriffe

- **Safety** (Sicherheit):
Daten werden nicht verändert
- **Idempotence**
(Idempotenz): Die Ressource behält auch bei mehrmaligen Aufruf den gleichen Zustand.

HTTP Method	Safe	Idempotent
GET		
POST		
PUT		
DELETE		
OPTIONS		
HEAD		

Weitere Informationen

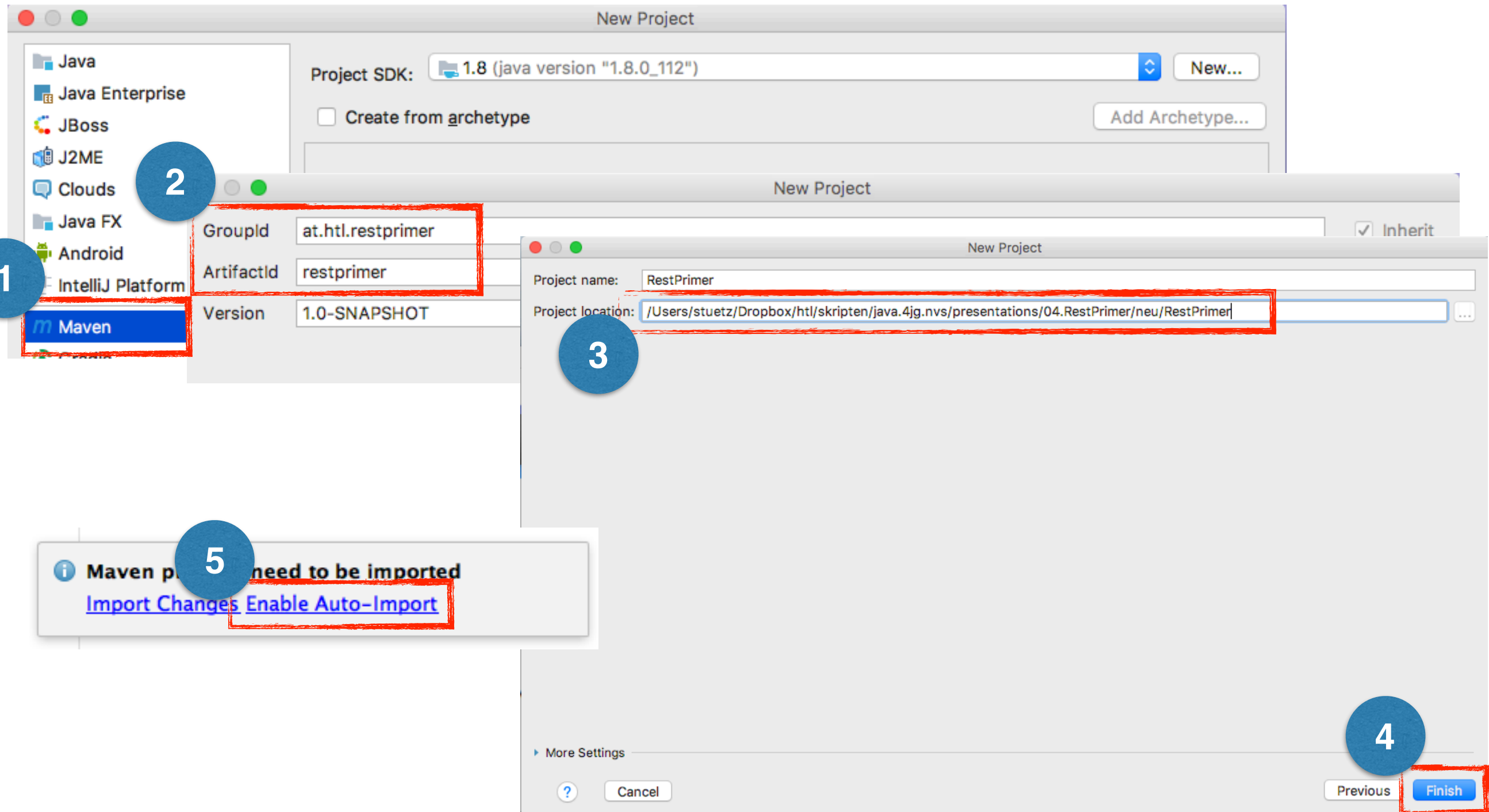


Beginning Java EE 7
1st Ed., 2013
Antonio Goncalves

RESTful API

GET PUT POST DELETE

Ein erstes Beispiel



pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

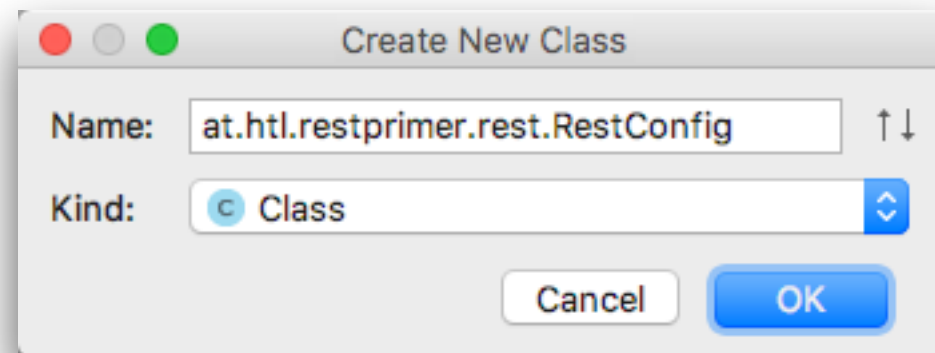
  <groupId>at.htl.restprimer</groupId>
  <artifactId>restprimer</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-web-api</artifactId>
      <version>7.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>restprimer</finalName>
  </build>
</project>
```

REST-Konfiguration



```
package at.htl.restprimer.rest;  
  
public class RestConfig extends App {  
    @ApplicationPath (javax.ws.rs)  
    Application (javafx.application)  
    Application (javax.faces.application)  
    Application (javax.ws.rs.core)  
    Application (com.apple.eawt)  
    Application (com.sun.glass.ui)  
    Application (com.sun.javaafx.tools.ant)
```

BEACHTET: Wir verwenden javax.ws.rs !!!

```
package at.htl.restprimer.rest;
```

```
import javax.ws.rs.ApplicationPath;  
import javax.ws.rs.core.Application;
```

```
@ApplicationPath("rs")  
public class RestConfig extends Application {  
}
```

Das ist die komplette Konfiguration, um REST-Ressourcen verwenden zu können. Ev. könnte man noch CORS konfigurieren. Warum wohl?

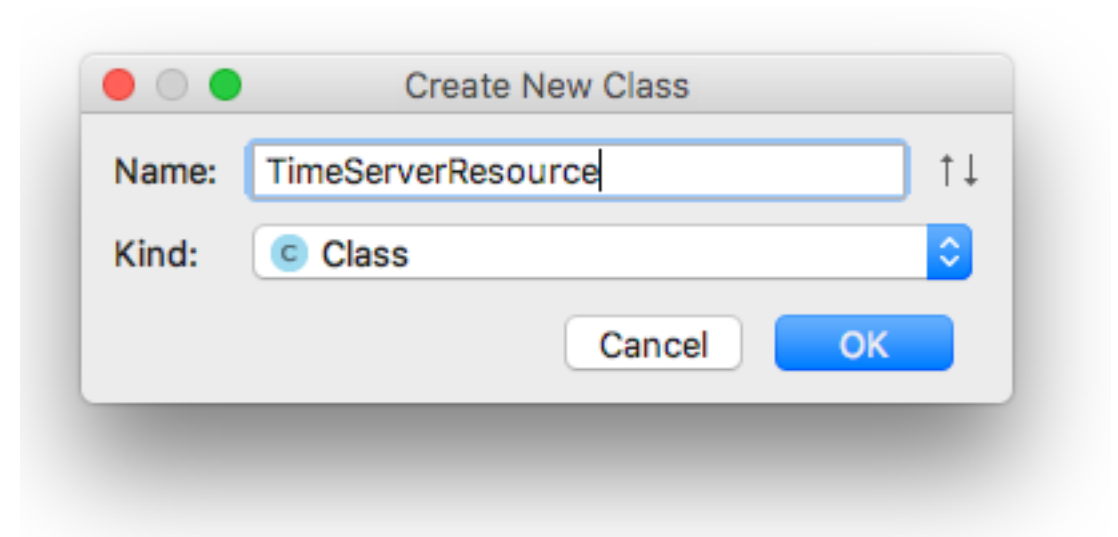
REST Ressource

```
package at.htl.restprimer.rest;

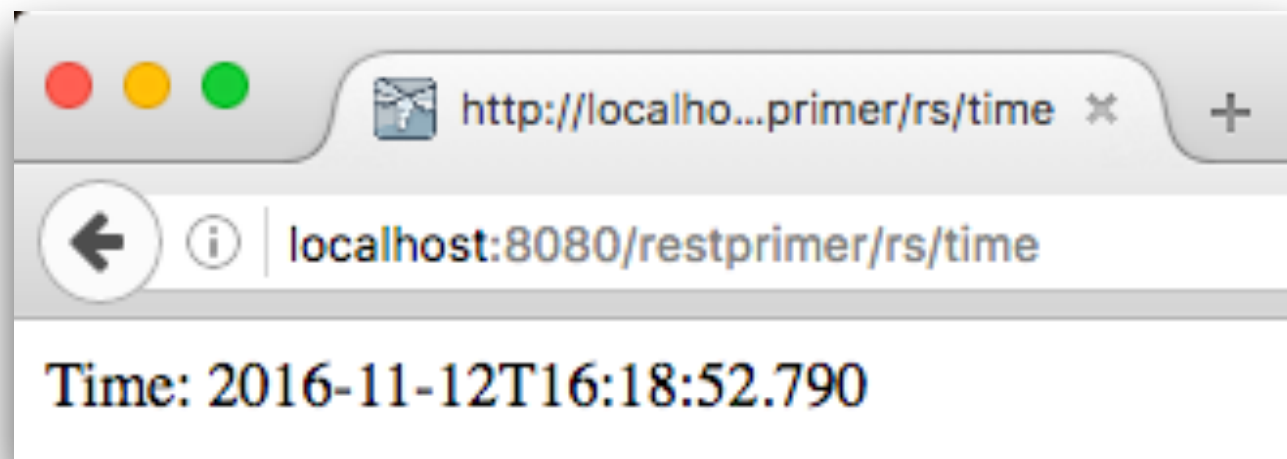
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import java.time.LocalDateTime;

@Path("/time")
public class TimeServerResource {

    @GET
    public String time() {
        return "Time: " + LocalDateTime.now();
    }
}
```



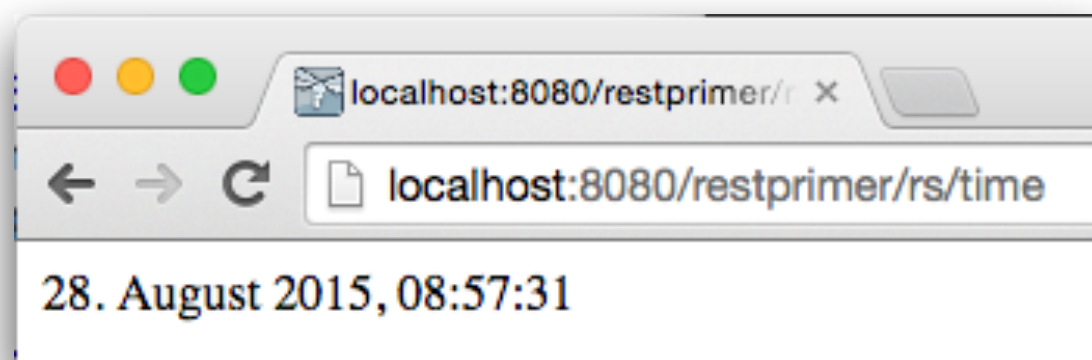
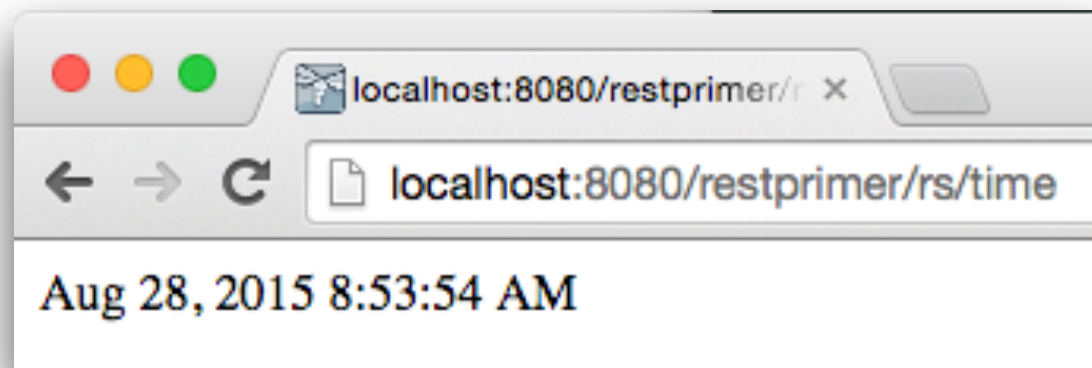
Testlauf



Nicht vergessen: Vor dem ersten Start den Application Server („Run/Debug“) konfigurieren

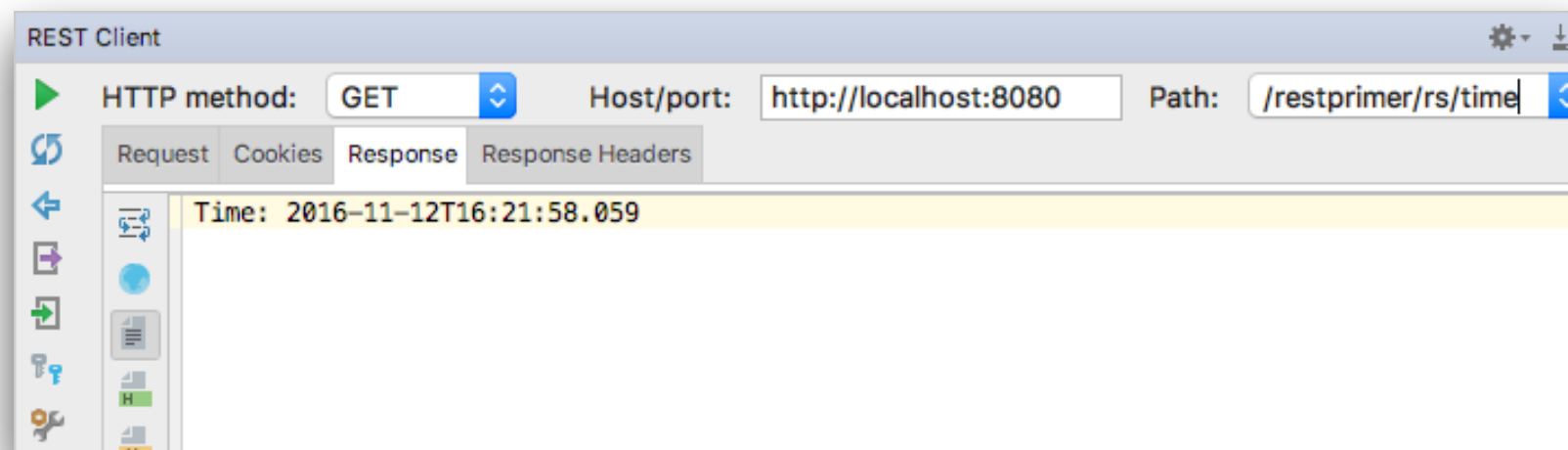
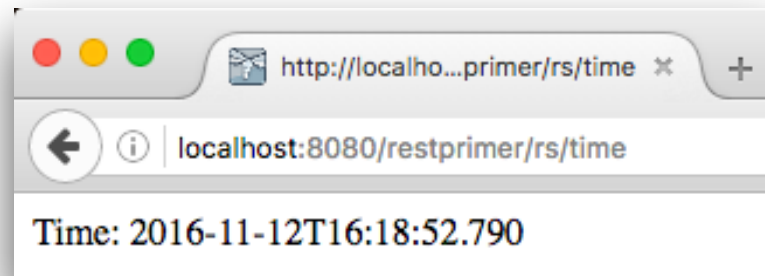
Verbesserung der Ausgabe

Wie muss hier die
Rückgabe aussehen?

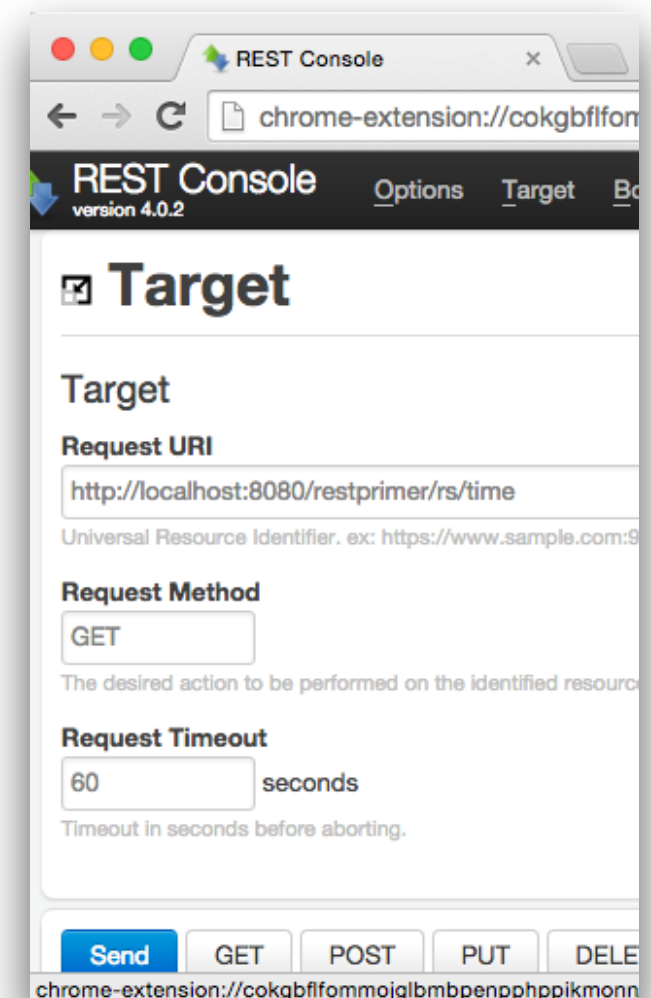
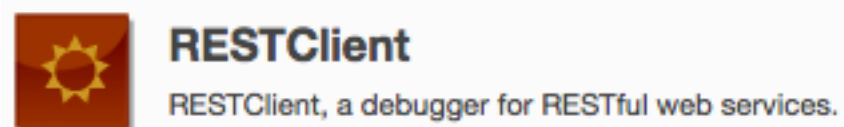


Mögliche Clients

- Web-Browser
- REST Client in IDE



- Plugins in WebBrowser
oder Standalone REST-Clients





File Authentication Headers View Favorite Requests Setting RESTClient

[-] Request

Method GET URL http://localhost:8080/restprimer/rs/time

☆ ▼

SEND

Body

Request Body

[-] Response

Response Headers

Response Body (Raw)

Response Body (Highlight)

Response Body (Preview)

28. August 2015, 09:10:38

1. Status Code : 200 OK

2. Connection : keep-alive

3. Content-Length : 25

4. Content-Type : text/html

5. Date : Fri, 28 Aug 2015 07:10:38 GMT

6. Server : WildFly/9

7. X-Powered-By : Undertow/1

Hier sieht man sehr gut die Header-Informationen wie Status Code und Content-Type

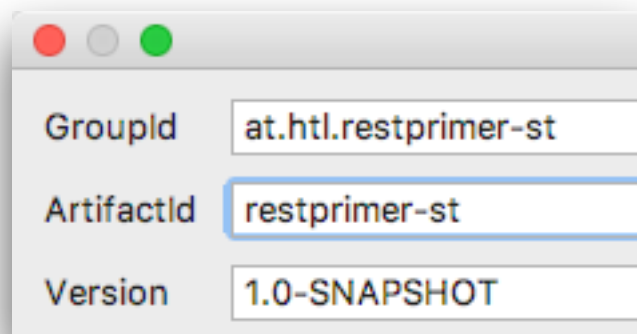
Eigener Java-Client

- Der Nachteil der vorher genannten Clients besteht darin, dass die REST-Abfragen i.N. nicht wiederholbar sind. D.h. bei jeder Änderung im Code, sind die Tests manuell durchzuführen
- Abhilfe: Die Erstellung eines eigenen Java-Clients (mit der Jersey API)

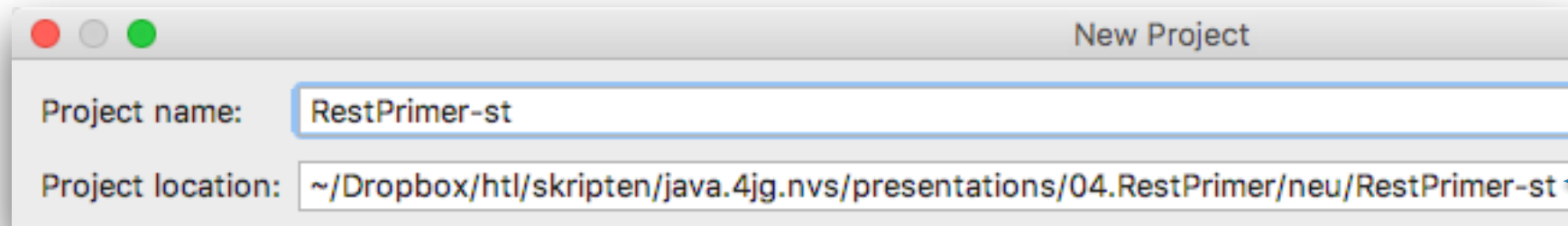
Exkurs: Java Community Process (JCP)

- Die Sprache Java wird durch den Java Community Process weiterentwickelt.
- Mitglieder des JCP (Firmen und Einzelpersonen) können Vorschläge zur Änderung der Sprache Java einbringen → Java Specification Request (JSR)
- Wird ein JSR angenommen, erstellt eine Expertengruppe eine Referenzimplementierung
- Für JAX-RS (Java API for RESTful Web Services) ist dies **Jersey**.
- Weitere Implementierungen sind zB RESTEasy, CXF, Restlet,...

Jersey-Testclient mit JUnit



GroupId at.htl.restprimer-st
ArtifactId restprimer-st
Version 1.0-SNAPSHOT

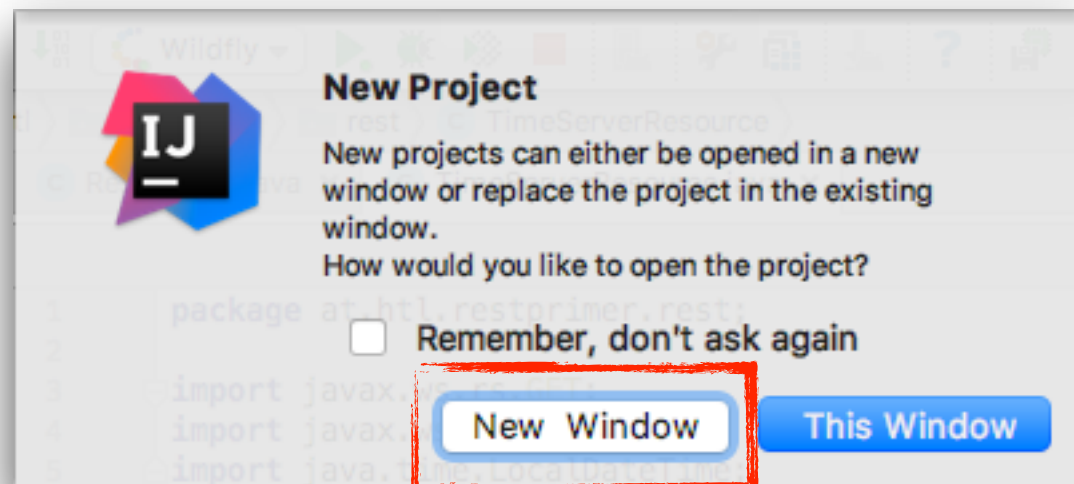


New Project

Project name: RestPrimer-st

Project location: ~/Dropbox/htl/skripten/java.4jg.nvs/presentations/04.RestPrimer/neu/RestPrimer-st

st ... Systemtest



New Project

New projects can either be opened in a new window or replace the project in the existing window.
How would you like to open the project?

☐ Remember, don't ask again

New Window This Window

Maven projects need to be imported
[Import Changes](#) [Enable Auto-Import](#)

<https://maven.apache.org/guides/mini/guide-naming-conventions.html>

pom.xml - 1 von 3

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>at.htl.restprimer-st</groupId>
  <artifactId>restprimer-st</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
```

pom.xml - 2 von 3

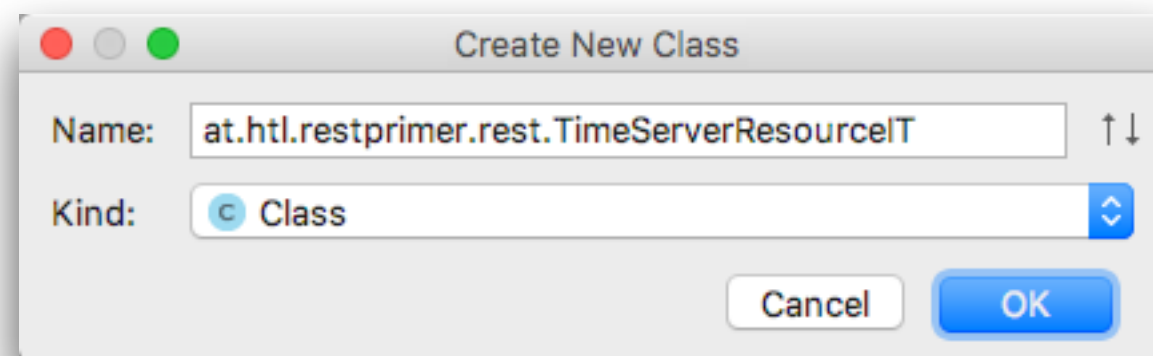
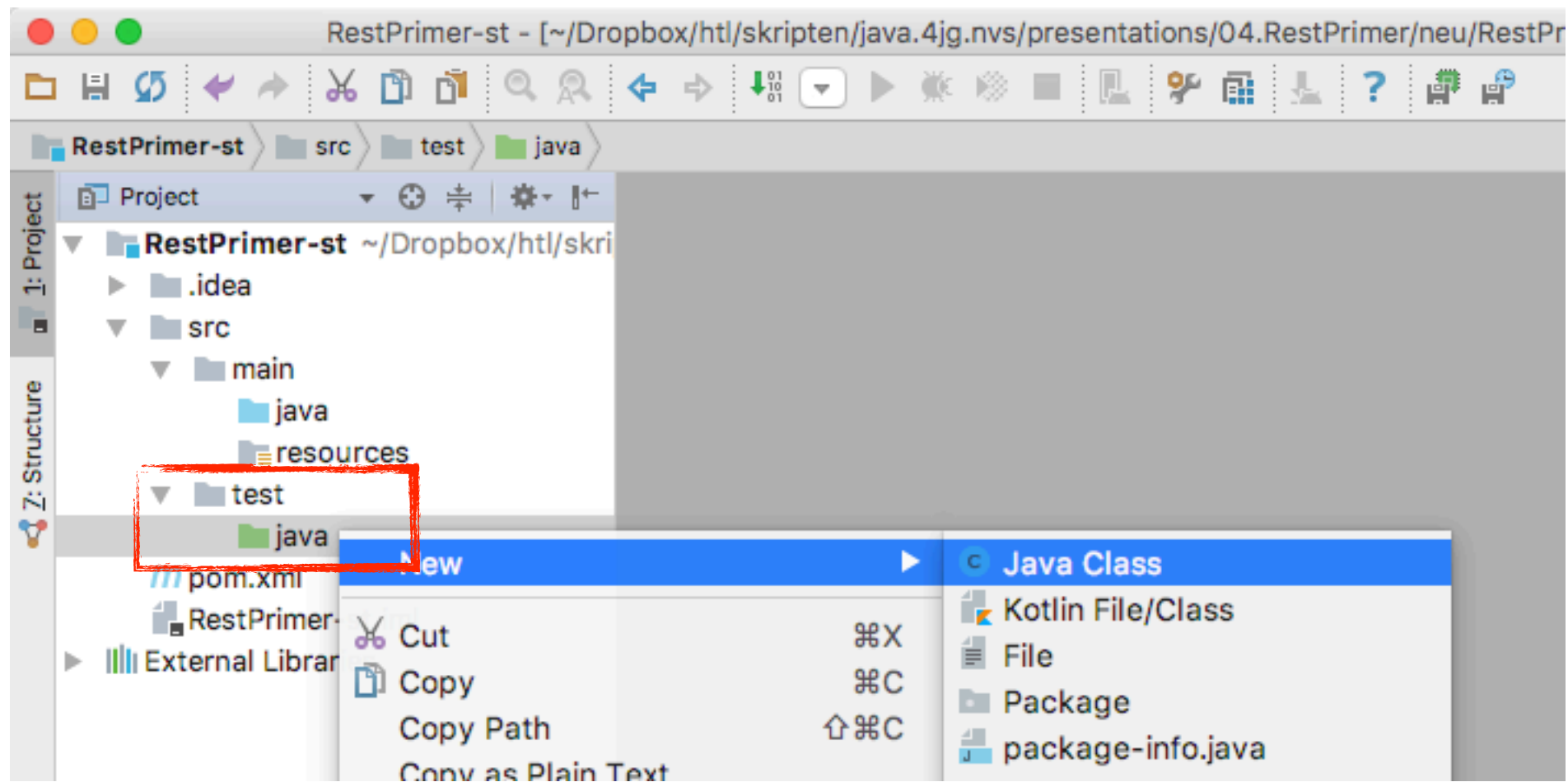
```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.core</groupId>
    <artifactId>jersey-client</artifactId>
    <version>2.27</version>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-processing</artifactId>
    <version>2.27</version>
  </dependency>
  <dependency>
    <groupId>org.glassfish</groupId>
    <artifactId>javax.json</artifactId>
    <version>1.1.2</version>
  </dependency>
</dependencies>
```

pom.xml - 3 von 3

```
<dependency>
  <groupId>org.glassfish.jersey.core</groupId>
  <artifactId>jersey-common</artifactId>
  <version>2.27</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.inject</groupId>
  <artifactId>jersey-hk2</artifactId>
  <version>2.27</version>
</dependency>
</dependencies>

</project>
```

Verwendet man eine jersey-Version \geq 2.26, so muss auch die Abhängigkeit für die Injection-Factory angegeben werden



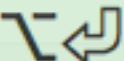
TimeServerResourceIT.java x

TimeServerResourceIT fetchTime()

```
1 package at.htl.restprimer.rest;
2
3 import org.junit.Before;
4 import org.junit.Test;
5
6 import javax.ws.rs.client.Client;
7 import javax.ws.rs.client.ClientBuilder;
8 import javax.ws.rs.client.WebTarget;
9 import javax.ws.rs.core.MediaType;
10 import javax.ws.rs.core.Response;
11
12 public class TimeServerResourceIT {
13
14     private Client client;
15     private WebTarget tut;
16
17     @Before
18     public void initClient() {
19         this.client = ClientBuilder.newClient();
20         this.tut = this.client.target(s: "http://localhost:8080/restprimer/rs/time");
21         // tut ... target under test
22     }
23
24     @Test
25     public void fetchTime() {
26         Response response = this.tut.request(MediaType.TEXT_PLAIN).get();
27         assertThat()~
28     }
29 }
30
```

IT ... IntegrationTest

Create method 'assertThat'
Import static method...

 (Alt+Enter for Win/Linux)

```

package at.htl.restprimer.rest;

import org.junit.Before;
import org.junit.Test;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.core.Is.is;
import static org.junit.Assert.assertTrue;

public class TimeServerResourceIT {

    private Client client;
    private WebTarget tut;

    @Before
    public void initClient() {
        this.client = ClientBuilder.newClient();
        this.tut = this.client.target("http://localhost:8080/restprimer/rs/time");
        // tut ... target under test
    }

    @Test
    public void fetchTime() {
        Response response = this.tut.request(MediaType.TEXT_PLAIN).get();
        assertThat(response.getStatus(), is(200));
        String payload = response.readEntity(String.class);
        assertTrue(payload.startsWith("Time: "));
    }
}

```



RestPrimer-st src test java at htl restprimer rest TimeServerResourceIT

Project TimeServerResourceIT.java x

RestPrimer-st ~/Dropbox/htl/skripten/ja

.idea

src

main

java

resources

test

java

at.htl.restprimer.rest

TimeServerResourceIT

pom.xml

RestPrimer-st.iml

External Libraries

Database

Maven Projects

Ant Build

Copy Reference ⌘⇧⌘C

Paste ⌘V

Paste from History... ⌘⇧V

Paste Simple ⌘⇧⌘V

Column Selection Mode ⌘⇧8

Refactor ▶

Folding ▶

Analyze ▶

Go To ▶

Generate... ⌘N

Recompile '...rverResourceIT.java' ⌘⇧F9

Run 'TimeServerResourceIT' ⌘⇧R

Debug 'TimeServerResourceIT' ⌘⇧D

Run 'TimeServerResourceIT' with Coverage

Create 'TimeServerResourceIT'...

Local History ▶

Compare with Clipboard

File Encoding

Create Gist...

WebServices ▶

1 package at.htl.restprimer.rest;

2

3 import ...

15

16 public class TimeServerResourceIT {

17

18 private Client client;

19 private WebTarget tut;

20

21 @Before

22 public void initClient() {

23 this.client = ClientBuilder.newClient();

24 this.tut = this.client.target(s: "http://localhost:8080/restprimer/rs/time");

25 // tut ... target under test

26 }

27

28 @Test

29 public void fetchTime() {

30 Response response = this.tut.request(MediaType.TEXT_PLAIN).get();

31 assertThat(response.getStatus(), is(value: 200));

32 String payload = response.readEntity(String.class);

33 assertTrue(payload.startsWith("Time: "));

34 }

35 }

36

in den weißen Bereich
außerhalb der Methoden
klicken, um „alle“ Tests
auszuführen

Man sollte jeden Test einmal
fehlschlagen lassen

The screenshot shows an IDE with the following components:

- Project Structure:** A tree view on the left showing the project hierarchy: `RestPrimer-st` (root) → `src` → `test` → `java` → `at.htl.restprimer.rest`. The file `TimeServerResourceIT` is selected.
- Code Editor:** Displays the source code of `TimeServerResourceIT.java`.

```
1 package at.htl.restprimer.rest;
2
3 import ...
4
15
16 public class TimeServerResourceIT {
17
18     private Client client;
19     private WebTarget tut;
20
21     @Before
22     public void initClient() {
23         this.client = ClientBuilder.newClient();
24         this.tut = this.client.target("http://localhost:8080/restprimer/rs/time");
25         // tut ... target under test
26     }
27
28     @Test
29     public void fetchTime() {
30         Response response = this.tut.request(MediaType.TEXT_PLAIN).get();
31         assertThat(response.getStatus(), is(value: 200));
32         String payload = response.readEntity(String.class);
33         assertTrue(payload.startsWith("Time: "));
34     }
35 }
36
```
- Run Console:** Shows the execution of the test. A progress bar indicates "1 test passed – 763ms". The test name is `TimeServerResour` (truncated) and the method is `fetchTime`, both taking 763ms. The output shows the Java command used and "Process finished with exit code 0".
- Bottom Bar:** Includes tabs for Terminal, Messages, Run, and TODO. A status bar at the bottom left says "Tests Passed: 1 passed (3 minutes ago)".

besser, da aussagekräftigere
Fehlermeldungen im Falle eines Fehlers
ist die Verwendung von hamcrest
`assertThat(payload, startsWith("Time"));`

Lösung

```
package at.htl.restprimer.rest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

@Path("/time")
public class TimeServerResource {

    @GET
    public String time() {
        return "Time: " + LocalDateTime
            .now()
            .format(DateTimeFormatter.ofPattern("dd. MMMM yyyy, hh:mm:ss"));
    }
}
```