

**SEW**

IT-Medientechnik

**NVS**

Informatik

# RESTful Services

mit Entities

# Projekt Vehicle - Server

```
package at.htl.vehicle.rest;

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

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

```
package at.htl.vehicle.rest;

import at.htl.vehicle.entity.Vehicle;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import java.util.ArrayList;
import java.util.List;

@Path("vehicle")
public class VehicleEndpoint {

    @GET
    @Path("{id}")
    public Vehicle find(@PathParam("id") long id) {
        return new Vehicle("Opel " + id, "Commodore");
    }

    @GET
    public List<Vehicle> findAll() {
        List<Vehicle> all = new ArrayList<>();
        all.add(find(42));
        return all;
    }
}
```

```
@XmlElement
package at.htl.vehicle.entity;

public class Vehicle {
    private String brand;
    private String type;

    public Vehicle() {}

    public Vehicle(String brand, String type) {
        this.brand = brand;
        this.type = type;
    }

    //region Getter and setter
    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public String getType() {
        return type;
    }

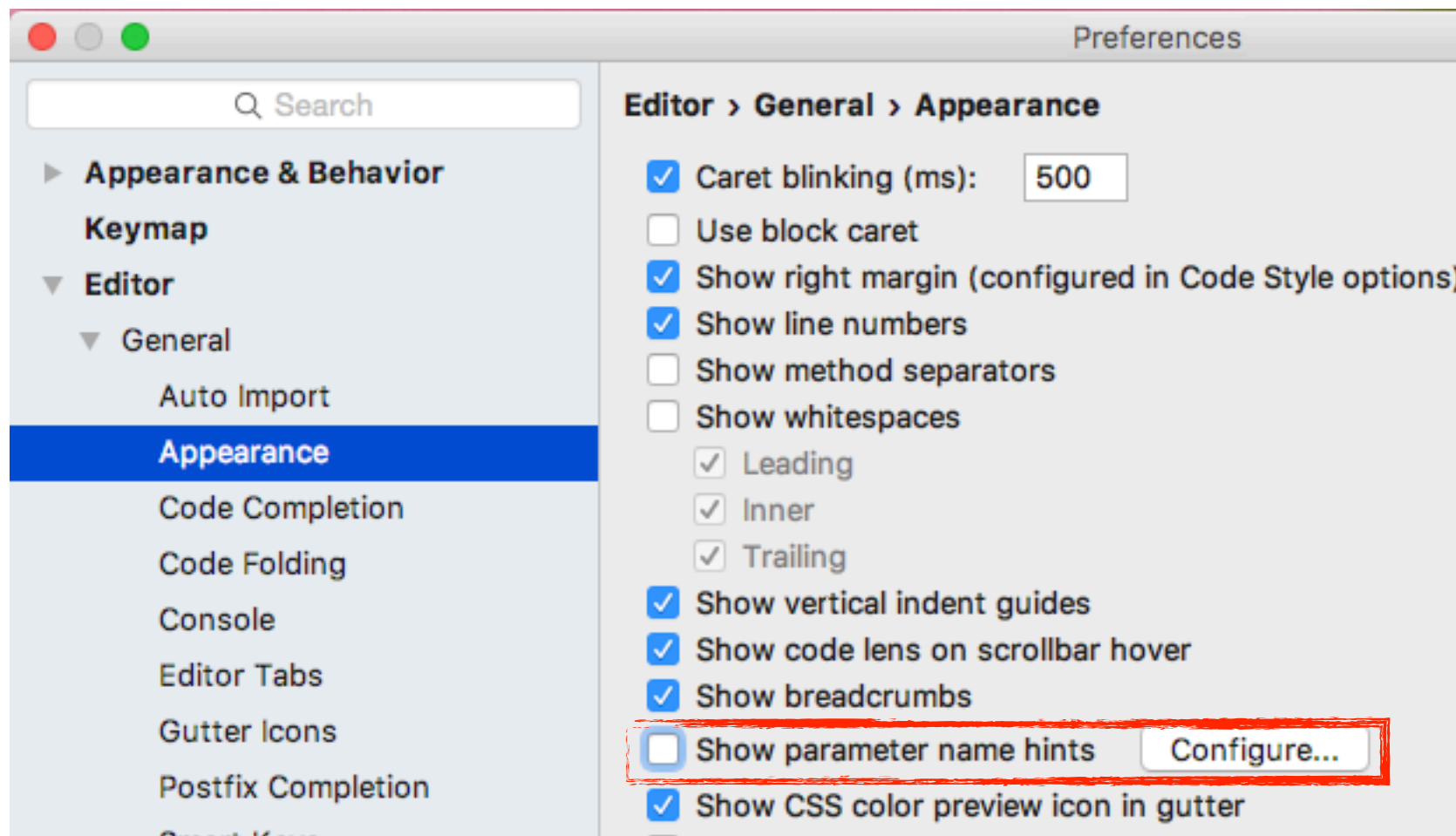
    public void setType(String type) {
        this.type = type;
    }

    //endregion
}
```

# Show parameter name hints

```
@GET
@Path("/{id}")
public Vehicle find(@PathParam("id") long id) {
    return new Vehicle(brand: "Opel " + id, type: "Commodore");
}
```

Manchmal irritieren diese Hints



```
@GET
@Path("/{id}")
public Vehicle find(@PathParam("id") long id) {
    return new Vehicle("Opel " + id, "Commodore");
}
```

# Testclient

### [-] Request

Method

URL

☆ ▼

**SEND**

### Body

Request Body

### [-] Response

Response Headers

Response Body (Raw)

Response Body (Highlight)

Response Body (Preview)

```
Could not find MessageBodyWriter for response object of type: at.htl.vehicle.entity.Vehicle of media type: text/html
```

### [-] Request

Method GET



URL http://localhost:8080/vehicle/rs/vehicle/1



SEND

### Body

Request Body

### [-] Response

Response Headers

Response Body (Raw)

Response Body (Highlight)

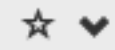
Response Body (Preview)

- 1. Status Code : 500 Internal Server Error
- 2. Connection : keep-alive
- 3. Content-Length : 116
- 4. Content-Type : text/html
- 5. Date : Sun, 13 Nov 2016 16:58:30 GMT
- 6. Server : WildFly/10
- 7. X-Powered-By : Undertow/1

### [-] Request

Method GET

URL



SEND

### Headers

Remove All

Accept: application/json ×

### Body

Request Body

### [-] Response

Response Headers

Response Body (Raw)

Response Body (Highlight)

Response Body (Preview)

```
1. Status Code      : 200 OK
2. Connection      : keep-alive
3. Content-Length  : 37
4. Content-Type    : application/json
5. Date            : Sun, 13 Nov 2016 17:00:32 GMT
6. Server          : WildFly/10
7. X-Powered-By   : Undertow/1
```

### [-] Request

Method  URL

### Headers

Remove All

Accept: application/json

### Body

Request Body

### [-] Response

Response Headers

Response Body (Raw)

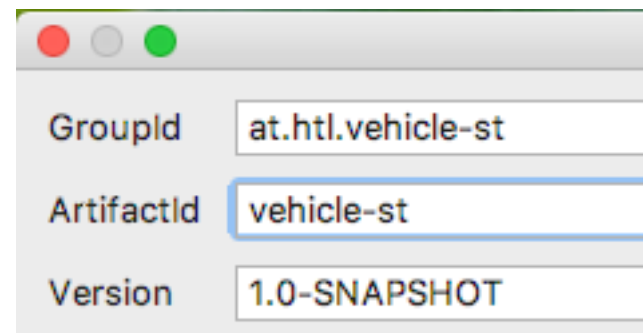
Response Body (Highlight)

Response Body (Preview)

```
{"brand": "Opel 1", "type": "Commodore"}
```

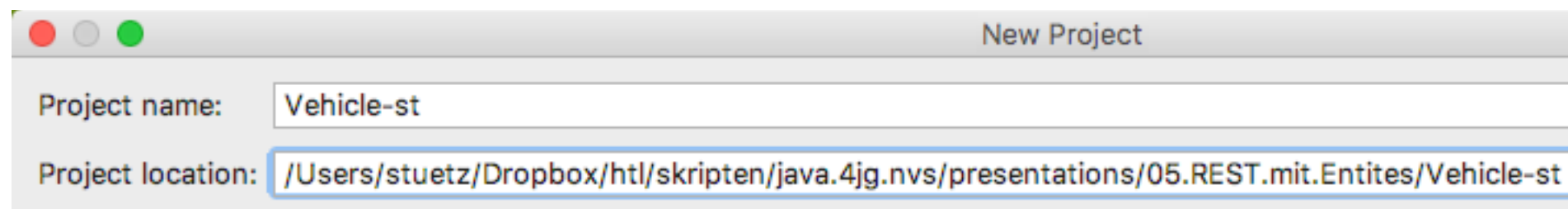


# jUnit-Client



A screenshot of a Maven configuration dialog box. It features three input fields: "GroupId" with the value "at.htl.vehicle-st", "ArtifactId" with the value "vehicle-st", and "Version" with the value "1.0-SNAPSHOT". The "ArtifactId" field is highlighted with a blue border.

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



A screenshot of a "New Project" dialog box. It features two input fields: "Project name" with the value "Vehicle-st" and "Project location" with the value "/Users/stuetz/Dropbox/htl/skripten/java.4jg.nvs/presentations/05.REST.mit.Entites/Vehicle-st". The "Project location" field is highlighted with a blue border.

Project name:	Vehicle-st
Project location:	/Users/stuetz/Dropbox/htl/skripten/java.4jg.nvs/presentations/05.REST.mit.Entites/Vehicle-st

```
<?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.vehicle-st</groupId>
  <artifactId>vehicle-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>
```

```
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.hamcrest</groupId>
          <artifactId>hamcrest-core</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.hamcrest</groupId>
      <artifactId>hamcrest-all</artifactId>
      <version>1.3</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish.jersey.core</groupId>
      <artifactId>jersey-client</artifactId>
      <version>2.24</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish.jersey.media</groupId>
      <artifactId>jersey-media-json-processing</artifactId>
      <version>2.24</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish</groupId>
      <artifactId>javax.json</artifactId>
      <version>1.1.3</version>
    </dependency>
  </dependencies>
```

```
</project>
```



# VehicleEndpointIT

```
package at.htl.vehicle.rest;

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

import javax.json.JsonArray;
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.CoreMatchers.*;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.junit.Assert.fail;

public class VehicleEndpointIT {
    private Client client;
    private WebTarget target;

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

    @Test
    public void fetchVehicle() {
        Response response = this.target.request(MediaType.TEXT_PLAIN).get();
        assertThat(response.getStatus(), is(200));
        String payload = response.readEntity(String.class);
        System.out.println("payload = " + payload);
    }
}
```

## Variante 1: TEXT\_PLAIN

The screenshot displays an IDE window with the following components:

- Project Explorer:** Shows the project structure for 'Vehicle-st' with folders for 'src/main/java', 'src/main/resources', and 'src/test/java'. The 'VehicleEndpointIT' class is highlighted in the 'test/java' folder.
- Code Editor:** Shows the source code for 'VehicleEndpointIT.java'.

```
26 @Before
27 public void initClient() {
28     this.client = ClientBuilder.newClient();
29     this.target = client.target("http://localhost:8080/vehicle/rs/vehicle");
30 }
31
32 @Test
33 public void fetchVehicle() {
34     Response response = this.target.request(MediaType.TEXT_PLAIN) get();
35     assertThat(response.getStatus(), is(200));
36     String payload = response.readEntity(String.class);
37     System.out.println("payload = " + payload);
38     fail();
39 }
40 }
```
- Run Console:** Shows the execution of the test. A red progress bar indicates '1 test failed - 704ms'. The output shows:

```
/Library/Java/JavaVirtualMachines/idk1.8.0_112.jdk/Contents/Home/bin/java ...
payload = [at.htl.vehicle.entity.Vehicle@1f9b0249]
java.lang.AssertionError <2 internal calls>
    at at.htl.vehicle.rest.VehicleEndpointIT.fetchVehicle(VehicleEndpointIT.java:38)
    <28 internal calls>
```
- Process Information:** At the bottom, it states 'Process finished with exit code 255'.

## Var. 2: APPLICATION\_JSON

The screenshot displays an IDE window for a project named "Vehicle-st". The main editor shows the file "VehicleEndpointIT.java" with the following code:

```
VehicleEndpointIT fetchVehicle()
26     @Before
27     public void initClient() {
28         this.client = ClientBuilder.newClient();
29         this.target = client.target("http://localhost:8080/vehicle/rs/vehicle");
30     }
31
32     @Test
33     public void fetchVehicle() {
34         Response response = this.target.request(MediaType.APPLICATION_JSON).get();
35         assertThat(response.getStatus(), is(200));
36         String payload = response.readEntity(String.class);
37         System.out.println("payload = " + payload);
38         fail();
39     }
40 }
```

The IDE's Run window shows the execution of the test. The output indicates that the test failed with an exit code of 255. The console output shows the following:

```
1 test failed - 719ms
VehicleEndpointIT (at. 719ms)
VehicleEndpointIT: 719ms
/Library/java/javaVirtualMachines/jdk1.8.0.112_jdk/Contents/Home/bin/java ...
payload = [{"brand":"Opel 42","type":"Commodore"}]
java.lang.AssertionError <2 internal calls>
    at at.htl.vehicle.rest.VehicleEndpointIT.fetchVehicle(VehicleEndpointIT.java:38)
    <28 internal calls>
Process finished with exit code 255
```



## Var. 3: APPLICATION\_XML

The screenshot displays an IDE window with a project named "Vehicle-st". The file explorer on the left shows the project structure, including a test directory with a "VehicleEndpointIT" class. The main editor shows the code for "VehicleEndpointIT.java", specifically the "fetchVehicle()" method. The code is as follows:

```
VehicleEndpointIT fetchVehicle()
26     @Before
27     public void initClient() {
28         this.client = ClientBuilder.newClient();
29         this.target = client.target("http://localhost:8080/vehicle/rs/vehicle");
30     }
31
32     @Test
33     public void fetchVehicle() {
34         Response response = this.target.request(MediaType.APPLICATION_XML).get();
35         assertThat(response.getStatus(), is(200));
36         String payload = response.readEntity(String.class);
37         System.out.println("payload = " + payload);
38         fail();
39     }
40 }
```

The execution output at the bottom shows the test results. The "fetchVehicle" test failed with an error message:

```
Run VehicleEndpointIT
1 test failed - 917ms
VehicleEndpointIT (at 917ms)
  fetchVehicle 917ms
    /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java ...
    payload = <?xml version="1.0" encoding="UTF-8"
    standalone="yes"?><collection><vehicle><brand>Opel
    42</brand><type>Commodore</type></vehicle></collection>
    java.lang.AssertionError <2 internal calls>
      at at.htl.vehicle.rest.VehicleEndpointIT.fetchVehicle(VehicleEndpointIT.java:38)
    <28 internal calls>
```

# Testen gegen JsonObject

```
Project ~/Dropbox/htl/skripte
├── Vehicle-st
│   ├── .idea
│   └── src
│       ├── main
│       │   ├── java
│       │   └── resources
│       └── test
│           └── java
│               └── at.htl.vehicle.rest
│                   └── VehicleEndpointIT.java
├── target
│   ├── pom.xml
│   └── Vehicle-st.iml
└── External Libraries
```

```
VehicleEndpointIT fetchVehicle()
26     @Before
27     public void initClient() {
28         this.client = ClientBuilder.newClient();
29         this.target = client.target("http://localhost:8080/vehicle/rs/vehicle");
30     }
31
32     @Test
33     public void fetchVehicle() {
34         Response response = this.target.request(MediaType.APPLICATION_JSON).get();
35         assertThat(response.getStatus(), is(200));
36         JSONArray payload = response.readEntity(JsonArray.class);
37         System.out.println("payload = " + payload);
38
39         JsonObject vehicle = payload.getJSONObject(0);
40         assertThat(vehicle.getString("brand"), is("Opel 42"));
41         assertThat(vehicle.getString("type"), is("Commodore"));
42     }
43 }
```

Run VehicleEndpointIT

1 test passed – 732ms

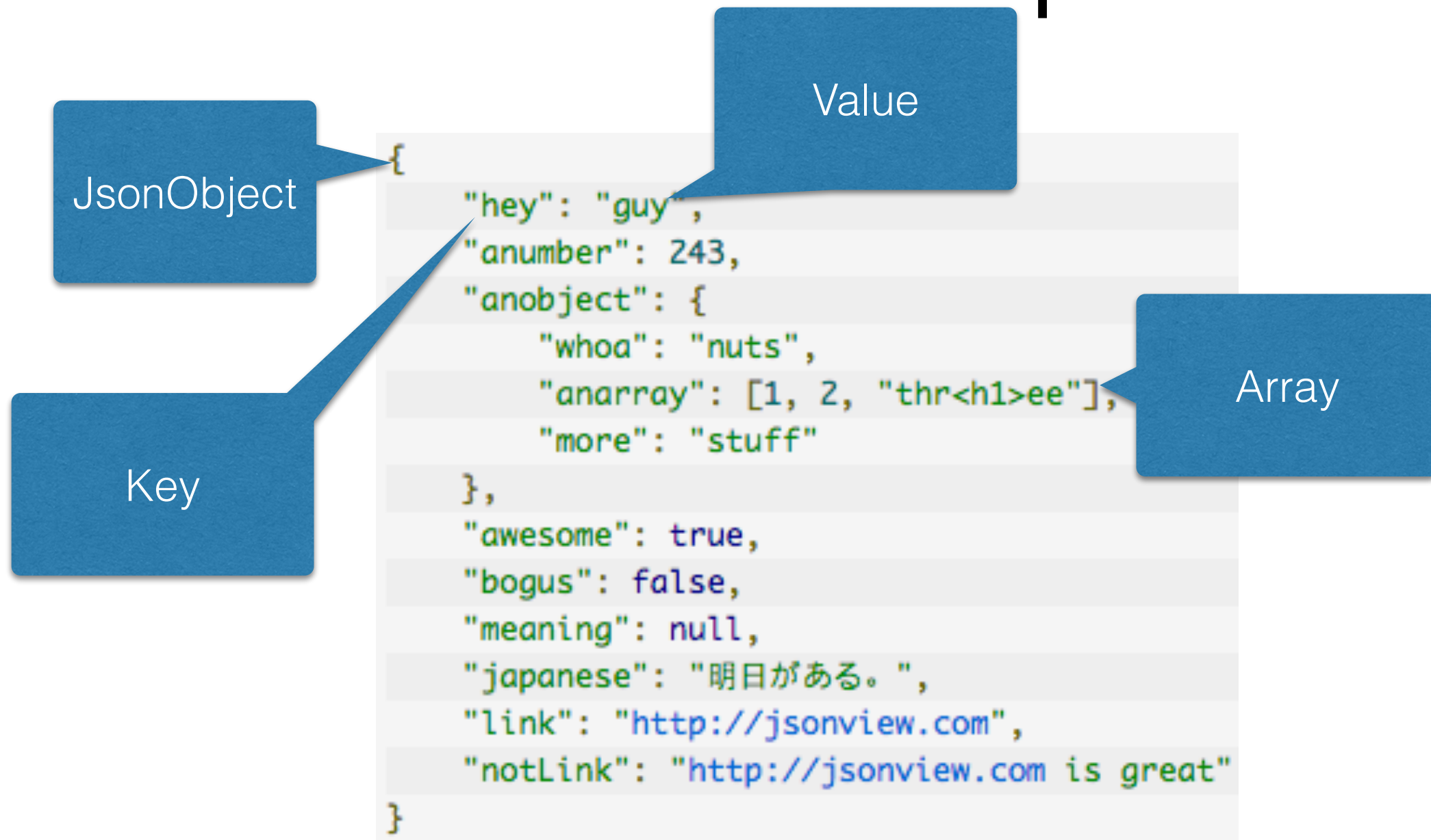
```
/Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java ...
payload = [{"brand":"Opel 42","type":"Commodore"}]
Process finished with exit code 0
```

# JSON


- JSON steht für "**JavaScript Object Notation**" und definiert ein Datenformat, in dem Informationen wie Objekte, Arrays und sonstige Variablen in lesbarer Form gespeichert werden können. In den meisten Sprachen gibt es Parser, die den JSON String in eine entsprechende Variable verwandeln.
- Die Notation von JSON in Java weicht in einigen Punkten von der JavaScript-Syntax ab:
  - Alle Eigenschaftsnamen in einem Objekt müssen in doppelten Anführungszeichen notiert sein.
  - Führende Kommas in Objekten und Arrays sind verboten.
  - Bei Zahlen sind führende Nullen verboten und einem Dezimalpunkt muss mindestens eine Ziffer folgen.
  - Strings müssen durch doppelte Anführungszeichen begrenzt sein. Es darf nur ein beschränktes Set von Zeichen escaped werden. Bestimmte Kontrollzeichen sind ganz verboten.
- JSON wird häufig in Verbindung mit Ajax genutzt, um einfach Informationen zwischen dem Clienten und dem Server auszutauschen und ist eine praktische Alternative zu XML.



# JSON Beispiel



# JSON Processing in Java

- JSON-P stellt zwei unterschiedliche JSON-APIs zur Verfügung:
- Object Model API: wir verwenden diese API 
- Streaming API: Low-Level-API, zur effizienten Verarbeitung großer JSON-Datenmengen, vergleichbar mit der StAX API für XML

# Erstellen eines JSON-Objekts

```
Employee emp = createEmployee();

JsonObjectBuilder empBuilder = Json.createObjectBuilder();
JsonObjectBuilder addressBuilder = Json.createObjectBuilder();
JsonArrayBuilder phoneNumBuilder = Json.createArrayBuilder();

for (long phone : emp.getPhoneNumbers()) {
    phoneNumBuilder.add(phone);
}

addressBuilder.add("street", emp.getAddress().getStreet())
               .add("city", emp.getAddress().getCity())
               .add("zipcode", emp.getAddress().getZipcode());

empBuilder.add("id", emp.getId())
           .add("name", emp.getName())
           .add("permanent", emp.isPermanent())
           .add("role", emp.getRole());

empBuilder.add("phoneNumbers", phoneNumBuilder);
empBuilder.add("address", addressBuilder);

JsonObject empJsonObject = empBuilder.build();

System.out.println("Employee JSON String\n"+empJsonObject);
```

# Zugriff auf JSON-Objekte

```
InputStream fis = new FileInputStream(JSON_FILE);

//create JsonReader object
JsonReader jsonReader = Json.createReader(fis);

//get JsonObject from JsonReader
JsonObject jsonObject = jsonReader.readObject();

//we can close IO resource and JsonReader now
jsonReader.close();
fis.close();
```

aus File in ein JsonObject

```
//Retrieve data from JsonObject and create Employee bean
Employee emp = new Employee();

emp.setId(jsonObject.getInt("id"));
emp.setName(jsonObject.getString("name"));
emp.setPermanent(jsonObject.getBoolean("permanent"));
emp.setRole(jsonObject.getString("role"));

//reading arrays from json
JsonArray jsonArray = jsonObject.getJsonArray("phoneNumbers");
long[] numbers = new long[jsonArray.size()];
int index = 0;
for(JsonValue value : jsonArray){
    numbers[index++] = Long.parseLong(value.toString());
}
emp.setPhoneNumbers(numbers);

//reading inner object from json object
JsonObject innerJsonObject = jsonObject.getJsonObject("address");
Address address = new Address();
address.setStreet(innerJsonObject.getString("street"));
address.setCity(innerJsonObject.getString("city"));
address.setZipcode(innerJsonObject.getInt("zipcode"));
emp.setAddress(address);

//print employee bean information
System.out.println(emp);
```

Zugriff auf Elemente des JsonObjects

# CRUD

Ein einfacher Einstieg



# Server - Entity

```
package at.htl.vehicle.entity;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Vehicle {

    private Long id;
    private String brand;
    private String type;

    public Vehicle() {}

    public Vehicle(String brand, String type) {
        this.brand = brand;
        this.type = type;
    }

    // Getter and setter
}
```

Mit dieser Annotation kann das Objekt in XML serialisiert werden

Die Entität wird mit einem künstlichen Schlüssel ergänzt

# Server - Endpoint

```
@Path("vehicle")
public class VehicleEndpoint {

    @GET
    @Path("{id}")
    public Vehicle find(@PathParam("id") long id) {
        return new Vehicle("Opel " + id, "Commodore");
    }

    @GET
    public List<Vehicle> findAll() {
        List<Vehicle> all = new ArrayList<>();
        all.add(find(42));
        return all;
    }

    @DELETE
    @Path("{id}")
    public void delete(@PathParam("id") long id) {
        System.out.println("deleted = " + id);
    }

    @POST
    public void save(Vehicle vehicle) {
        System.out.println("Vehicle = " + vehicle);
    }
}
```

# Client - Tests

```
public class VehicleEndpointIT {
    private Client client;
    private WebTarget target;

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

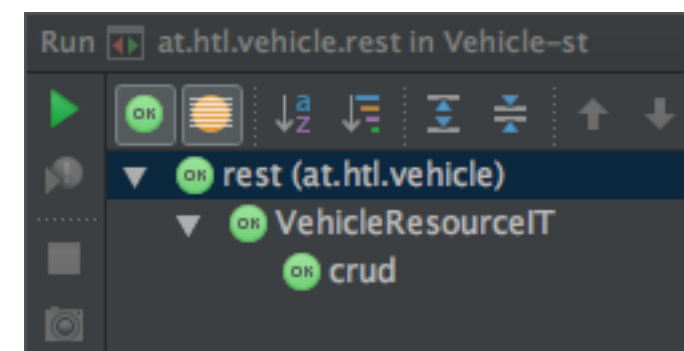
    @Test
    public void crud() {
        Response response = this.target.request(MediaType.APPLICATION_JSON).get();
        assertThat(response.getStatus(), is(200));
        JSONArray payload = response.readEntity(JsonArray.class);
        System.out.println("payload = " + payload);
        assertThat(payload, not(empty()));

        JsonObject vehicle = payload.getJsonObject(0);
        assertThat(vehicle.getString("brand"), equalTo("Opel 42"));
        assertThat(vehicle.getString("type"), startsWith("Commodore"));

        // GET with id
        JsonObject dedicatedVehicle = this.target
            .path("43")
            .request(MediaType.APPLICATION_JSON)
            .get(JsonObject.class);
        assertThat(dedicatedVehicle.getString("brand"), containsString("43"));
        assertThat(dedicatedVehicle.getString("brand"), equalTo("Opel 43"));

        Response deleteResponse = this.target
            .path("42")
            .request(MediaType.APPLICATION_JSON)
            .delete();
        assertThat(deleteResponse.getStatus(), is(204)); // no content
    }
}
```

- Diese Tests sind nicht optimal, da mehrere Testfälle in einer Testmethode enthalten sind.
- Die einzelnen Testfälle sollten voneinander unabhängig sein





```

package at.htl.vehicle.rest;

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

import javax.json.JsonArray;
import javax.json.JsonObject;
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.CoreMatchers.*;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.empty;
import static org.hamcrest.Matchers.isEmptyString;
import static org.hamcrest.core.IsNot.not;

public class VehicleEndpointIT {
    private Client client;
    private WebTarget target;

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

    @Test
    public void crud() {
        Response response = this.target.request(MediaType.APPLICATION_JSON).get();
        assertThat(response.getStatus(), is(200));
        JsonArray allTodos = response.readEntity(JsonArray.class);
        System.out.println("payload = " + allTodos);
        assertThat(allTodos, not(empty()));

        JsonObject vehicle = allTodos.getJsonObject(0);
        assertThat(vehicle.getString("brand"), equalTo("Opel 42"));
        assertThat(vehicle.getString("type"), startsWith("Commodore"));

        // GET with id
        JsonObject dedicatedVehicle = this.target
            .path("43")
            .request(MediaType.APPLICATION_JSON)
            .get(JsonObject.class);
        assertThat(dedicatedVehicle.getString("brand"), containsString("43"));
        assertThat(dedicatedVehicle.getString("brand"), equalTo("Opel 43"));

        Response deleteResponse = this.target
            .path("42")
            .request(MediaType.APPLICATION_JSON)
            .delete();
        assertThat(deleteResponse.getStatus(), is(204)); // no content
    }
}

```