RESTful Services

mit JPA

Projekt erstellen

- Ausgangspunkt für die nächsten Erweiterungen ist das Projekt „Vehicle“
- Öffnen Sie dieses.

Was ist zu tun?

- Status Quo: Wir haben nun eine einfache REST-Resource und eine Entität Vehicle.
- Das Ziel ist es einerseits die
  - Entitäten in der Datenbank zu persistieren, andererseits die
  - Geschäftslogik (Business Logic) in eigene Klassen auszulagern, um so die Testbarkeit zu verbessern
  - Diese Klasse nennen wir VehicleFacade

Auslagern der Business Logik

Erstellen einer Facade

Project | Packages

- Vehicle (~/Dropbox/htl/skripten/java.4jg.n
  - .idea
  - src
    - main
      - java
        - at.htl.vehicle
          - business
            - VehicleFacade
          - entity
            - Vehicle
          - rest
            - RestConfig
            - VehicleResource
      - resources
    - test

VehicleFacade.java

```java
package at.htl.vehicle.business;

import javax.ejb.Stateless;

@Stateless
public class VehicleFacade {
}
```

Derzeitige VehicleResource

```java
@Path("vehicle")
public class VehicleResource {

    @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);
    }
```

Auslagern der Business Logik

```java
@Stateless
public class VehicleFacade {

    public Vehicle findById(long id) {
        return new Vehicle("Opel " + id,"Commodore");
    }

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

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

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

Die Vehicle-Resource vereinfacht sich

```java
@Stateless
@Path("vehicle")
public class VehicleResource {

    @Inject
    VehicleFacade vehicleFacade;

    @GET
    @Path("{id}")
    public Vehicle find(@PathParam("id") long id) {
        return vehicleFacade.findById(id);
    }

    @GET
    public List<Vehicle> findAll() {
        return vehicleFacade.findAll();
    }

    @DELETE
    @Path("{id}")
    public void delete(@PathParam("id") long id) {
        vehicleFacade.delete(id);
    }

    @POST
    public void save(Vehicle vehicle) {
        this.vehicleFacade.save(vehicle);
    }
}
```

Die Business Logik wird entfernt, die REST-Resource besteht eigentlich „nur“ noch aus Aufrufen der Business-Logik.
Die Schnittstelle nach außen hat sich nicht verändert. Alle Tests funktionieren noch.

Konfigurieren von JPA mit JBoss Forge

http://forge.jboss.org/download

Show Forge Commands via ⌥⌘4 (Ctrl+Alt+4 for Win/Linux)

JPA: Setup [at.htl.vehicle.entity.Vehicle]

Setup JPA in your project

JPA Version: 2.1

Container: Wildfly Application Server

Provider: Hibernate

☐ Install a JPA 2 metamodel generator?

- Hibernate 4.x
- OpenJPA
- Infinispan
- Java EE
- ✓ Hibernate
- Eclipse Link

< Previous | Next > | Finish | Cancel | Help

Hier ist es vorteilhafter „Java EE“ anstelle von Hibernate zu wählen, da dann in der persistence.xml die Standard-JPA-Poperties verwendet werden.

JPA: Setup [at.htl.vehicle.entity.Vehicle]

Configure your connection settings

Persistence Unit Name: myPU

Database Type: DERBY

Overwrite Persistence Unit

DataSource Name: java:jboss/datasources/DbDS

Schema Generation Type: (•) Drop and Create ( ) Drop ( ) Create ( ) None

< Previous | Next > | Finish | Cancel | Help

persistence.xml

Command executed successfully

File is not configured as JPA facet des

Frameworks detected
JPA framework is detected in the project Configure

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/200
3    <persistence-unit name="myPU" transaction-type="JTA">
4      <description>Forge Persistence Unit</description>
5      <provider>org.hibernate.ejb.HibernatePersistence</provider>
6      <jta-data-source>java:jboss/datasources/DsDS</jta-data-source>
7      <exclude-unlisted-classes>false</exclude-unlisted-classes>
8      <properties>
9        <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
10       <property name="hibernate.show_sql" value="true"/>
11       <property name="hibernate.format_sql" value="true"/>
12       <property name="hibernate.transaction.flush_before_completion" value="true"/>
13       <property name="hibernate.dialect" value="org.hibernate.dialect.DerbyDialect"/>
14     </properties>
15   </persistence-unit>
16 </persistence>
17
```

JPA im IntelliJ-Projekt konfigurieren

```
<property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
```

„javax.persistence"-Property im Gegensatz zu „hibernate"-Properties

Setup Frameworks

Group by: type

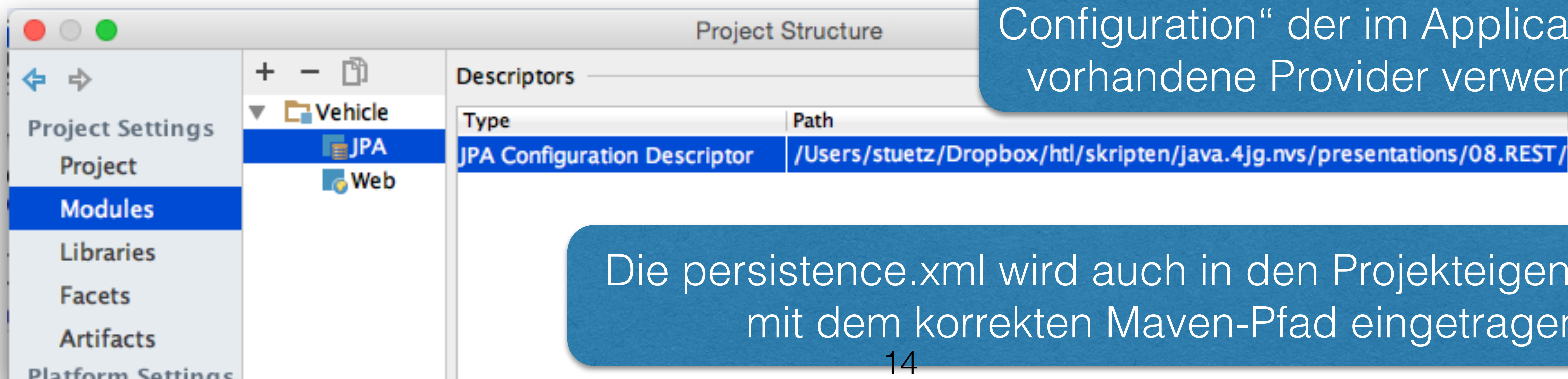
- [x] JPA
  - [x] persistence.xml (/Users/stuetz/Dropbox/htl/skripten/java.4jg.nvs/presentations/08.REST/Vehicle/src/main/resources/META-INF)

Cancel OK

persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.c
  <persistence-unit name="myPU" transaction-type="JTA">
    <jta-data-source>java:jboss/datasources/DsDS</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.transaction.flush_before_completion" value="true"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.DerbyDialect"/>
    </properties>
  </persistence-unit>
</persistence>
```

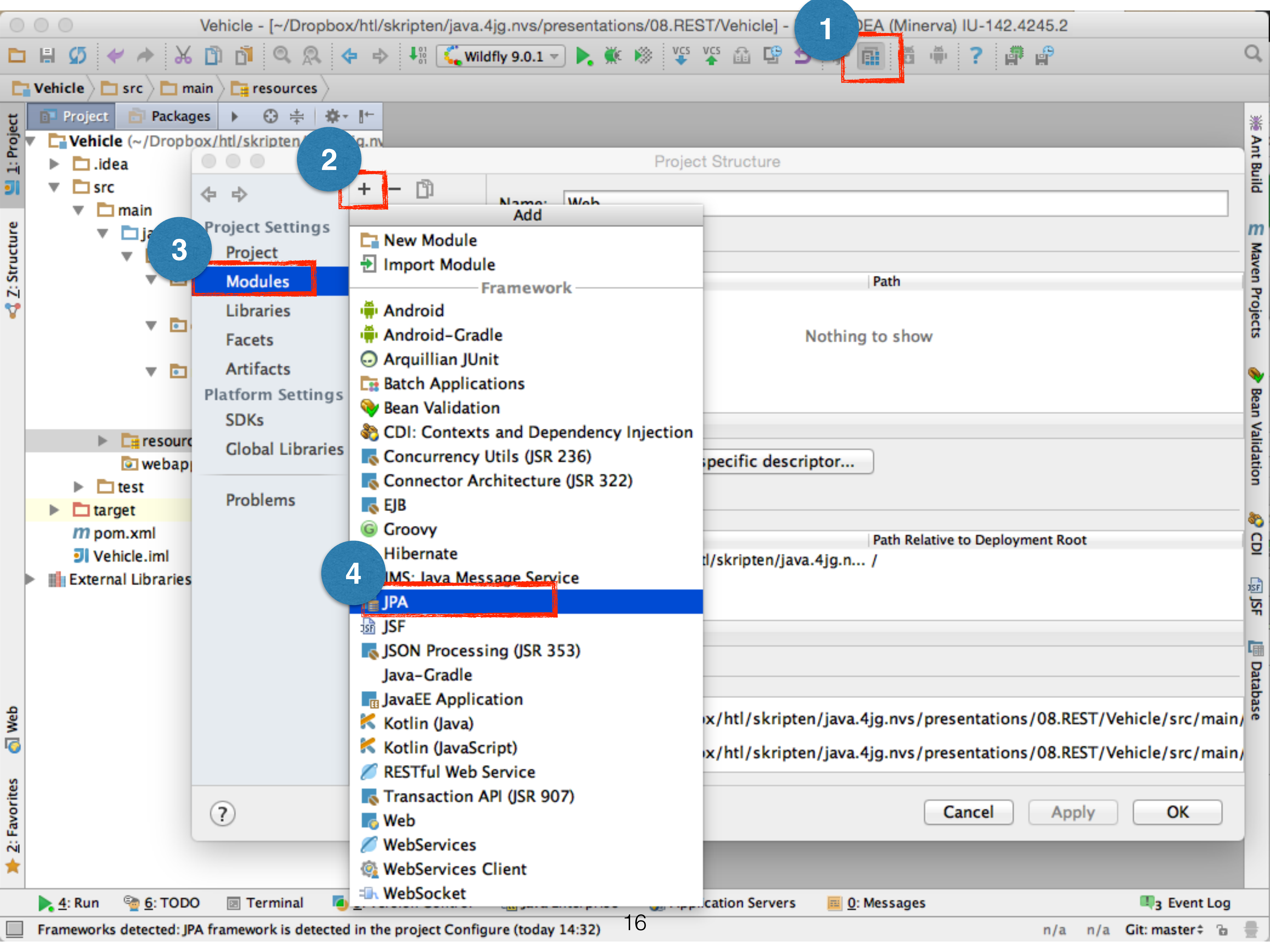
Der <provider>-Tag wird gelöscht, da wegen „Convention over Configuration“ der im Application Server vorhandene Provider verwendet wird.



Die persistence.xml wird auch in den Projekteigenschaften mit dem korrekten Maven-Pfad eingetragen.

Konfigurieren von JPA in IntelliJ IDEA

alternativ zur Vorgangsweise mit JBoss Forge



Vehicle - [~/Dropbox/htl/skripten/java.4jg.nvs/presentations/08.REST/Vehicle] - IntelliJ IDEA (Minerva) IU-142.4245.2
1
2
3
4
Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
Add
New Module
Import Module
Framework
Android
Android-Gradle
Arquillian JUnit
Batch Applications
Bean Validation
CDI: Contexts and Dependency Injection
Concurrency Utils (JSR 236)
Connector Architecture (JSR 322)
EJB
Groovy
Hibernate
JMS: Java Message Service
JPA
JSF
JSON Processing (JSR 353)
Java-Gradle
JavaEE Application
Kotlin (Java)
Kotlin (JavaScript)
RESTful Web Service
Transaction API (JSR 907)
Web
WebServices
WebServices Client
WebSocket
Nothing to show
Cancel
Apply
OK
Frameworks detected: JPA framework is detected in the project Configure (today 14:32)

Vehicle - [~/Dropbox/htl/skripten/java.4jg.nvs/presentations/08.REST/Vehicle] - IntelliJ IDEA (Minerva) IU-142.4245.2

Vehicle › src › main › resources

Project | Packages

Vehicle (~/Dropbox/htl/skripten/java.4jg.nv
.idea
src
main
java
at.h
resourc
webap
test
target
pom.xml
Vehicle.iml
External Libraries

Project Structure

Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems

Vehicle
JPA
Web

Descriptors

Type	Path
JPA Configuration Descriptor	/Users/stuetz/Dropbox/htl/skripten/java.4jg.nvs/

Nothing to show

Deployment Descriptor Location

JPA Configuration Descriptor (persistence.xml):

/Vehicle/src/main/resources/META-INF/persistence.xml

Deployment descriptor version 2.1

Cancel | OK

1 persistence.xml
2 orm.xml

<no provider>

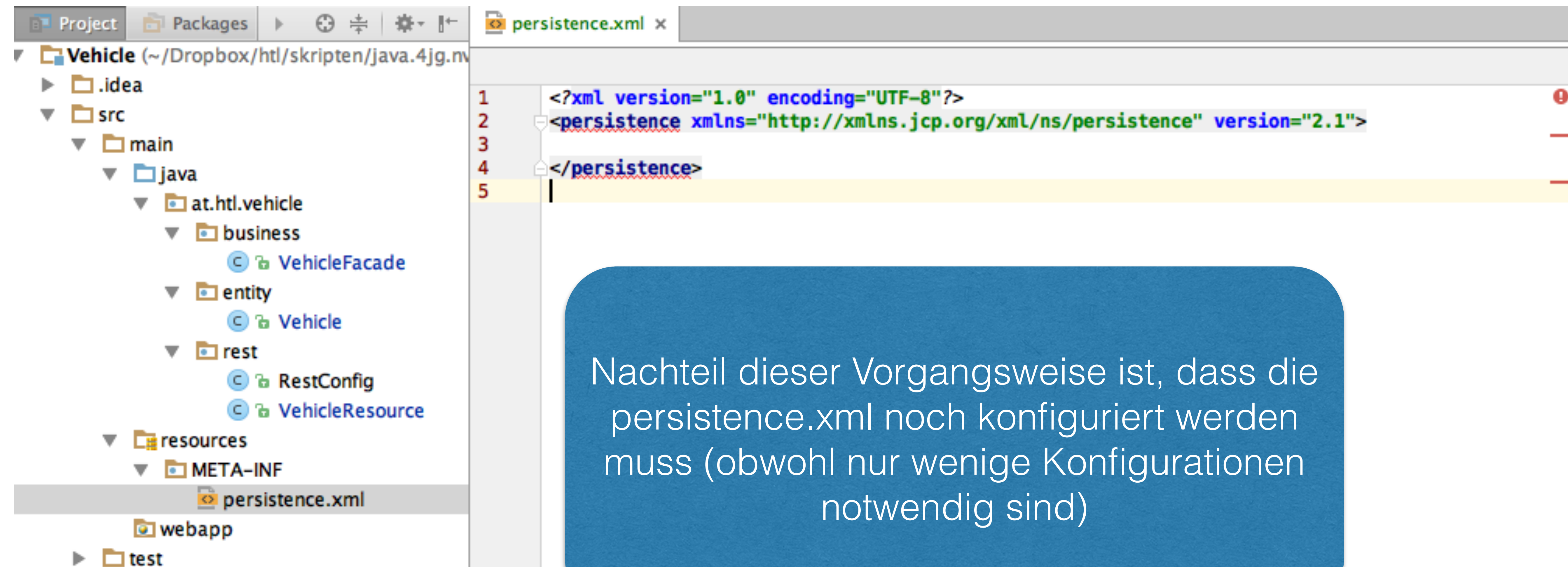
Cancel | Apply | OK

4: Run | 6: TODO | Terminal | 9: Version Control | Java Enterprise | Application Servers | 0: Messages | 3 Event Log

Frameworks detected: JPA framework is detected in the project Configure (today 14:32)

n/a n/a Git: master

persistence.xml



Minimale persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.1">
    <persistence-unit name="myPU" transaction-type="JTA">
    <jta-data-source>java:jboss/datasources/DbDS</jta-data-source>
        <properties>
            <property name="javax.persistence.schema-generation.database.action"
                      value="drop-and-create"/>
        </properties>
    </persistence-unit>
</persistence>
```

Selbst die <jta-data-source> müsste nicht angegeben werden, wenn im Application Server nur eine Datasource verfügbar ist.
Oft werden anstelle der JPA-Properties implementierungs-spezifische Properties verwendet, da mehr Einstellungsmöglichkeiten vorhanden sind.

DerbyDb starten

Terminal (in IntelliJ) öffnen

Terminal

```
Toms-MacBook-Pro:Vehicle stuetz$ pwd
/Users/stuetz/Dropbox/htl/skripten/java.4jg.nvs/presentations/08.REST/Vehicle
Toms-MacBook-Pro:Vehicle stuetz$ mkdir db
Toms-MacBook-Pro:Vehicle stuetz$ cd db
Toms-MacBook-Pro:db stuetz$ pwd
/Users/stuetz/Dropbox/htl/skripten/java.4jg.nvs/presentations/08.REST/Vehicle/db
Toms-MacBook-Pro:db stuetz$ $JAVA_HOME/db/bin/startNetworkServer -noSecurityManager
Wed Sep 02 20:01:10 CEST 2015 : Apache Derby Network Server - 10.11.1.2 - (1629631) started
```

Persistieren eines Objekts per REST Request

Vehicle.java

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

import ...

@Entity
@NamedQuery(name = "Vehicle.findAll", query = "SELECT v FROM Vehicle v")
@XmlRootElement
public class Vehicle {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    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
}
```

VehicleFacade.java

```java
@Stateless
public class VehicleFacade {

    @PersistenceContext
    EntityManager em;

    public Vehicle findById(long id) {
        return this.em.find(Vehicle.class, id);
    }

    public void delete(long id) {
        Vehicle reference = this.em.getReference(Vehicle.class, id);
        this.em.remove(reference);
    }

    public List<Vehicle> findAll() {
        return this.em
                .createNamedQuery("Vehicle.findAll", Vehicle.class)
                .getResultList();
    }

    public void save(Vehicle vehicle) {
        this.em.merge(vehicle);
    }
}
```

performanter als find()

VehicleResource.java

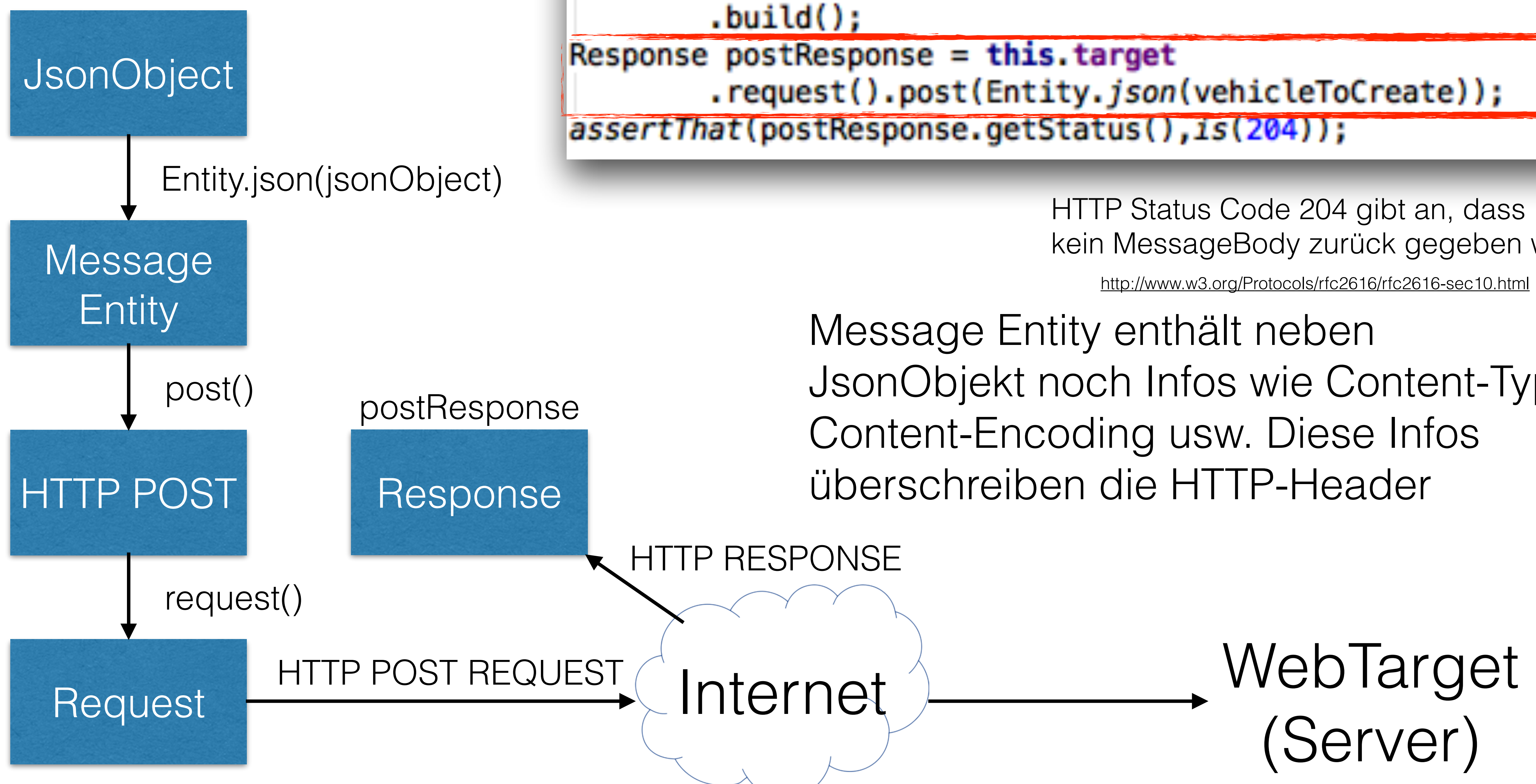
```
@Stateless
@Path("vehicle")
public class VehicleResource {

    @Inject
    VehicleFacade vehicleFacade;

    @GET
    @Path("{id}")
    public Vehicle find(@PathParam("id") long id) {
        return vehicleFacade.findById(id);
    }

    @GET
    public List<Vehicle> findAll() {
        return vehicleFacade.findAll();
    }

    @DELETE
    @Path("{id}")
    public void delete(@PathParam("id") long id) {
        vehicleFacade.delete(id);
    }

    @POST
    public void save(Vehicle vehicle) {
        this.vehicleFacade.save(vehicle);
    }
}
```

Test-Client

Message Entity

Jersey Client

```java
JsonObjectBuilder vehicleBuilder = Json.createObjectBuilder();
JsonObject vehicleToCreate = vehicleBuilder
        .add("brand", "Opel")
        .add("type", "Commodore")
        .build();
Response postResponse = this.target
        .request().post(Entity.json(vehicleToCreate));
assertThat(postResponse.getStatus(),is(204));
```



HTTP Status Code 204 gibt an, dass kein MessageBody zurück gegeben wurde

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Message Entity enthält neben JsonObjekt noch Infos wie Content-Type, Content-Encoding usw. Diese Infos überschreiben die HTTP-Header

Test-Client

Lesen aller Entites

```
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()));
```

Ausgabe in der Test-Console

```
payload = [{"id":1,"brand":"Opel","type":"Commodore"}]
```

Inhalt der Datenbank

	ID	BRAND	TYPE
1	1	Opel	Commodore

Erstellen einer korrekten Response beim Erstellen einer Ressource

Probleme

- Die save()-Methode in der VehicleResource sollte einen Rückgabe wert zurückgeben

- Vorgaben in RFC 2616

9.5 POST

The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

. . .

If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header (see section 14.30).

. . .

http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

- Wir werden daher die Methode save() , sowohl in VehicleFacade als auch VehicleResource

Methoden save()

VehicleFacade.java

```java
        long
public Vehicle save(Vehicle vehicle) {
    return this.em.merge(vehicle);
}                                    .getId();
```

VehicleResource.java

Der Pfad wird injiziert

```java
@POST
public Response save(Vehicle vehicle, @Context UriInfo info) {
    Vehicle saved = this.vehicleFacade.save(vehicle);
    Long id = saved.getId();
    URI uri = info.getAbsolutePathBuilder().path("/" + id).build();
    return Response.created(uri).build();
}
```

Test-Client

Testen der Response

```
JsonObjectBuilder vehicleBuilder = Json.createObjectBuilder();
JsonObject vehicleToCreate = vehicleBuilder
        .add("brand", "Opel")
        .add("type", "Commodore")
        .build();
Response postResponse = this.target
        .request().post(Entity.json(vehicleToCreate));
assertThat(postResponse.getStatus(),is(201));
String location = postResponse.getHeaderString("Location");
System.out.println("location = " + location);
```

nun wird der geforderte Status zurück gegeben

Ausgabe in der Test-Console

```
location = http://localhost:8080/vehicle/rs/vehicle/1
```

Der vollständige Pfad (inkl. Id) zur neu erstellten Ressource wird zurückgegeben

Test-Client

Test: Abfrage nach Id

Wir ändern nun die Abfrage anhand der Id und verwenden die beim POST retournierte location

```
// GET with id
JsonObject dedicatedVehicle = this.client
        .target(location)
        .request(MediaType.APPLICATION_JSON)
        .get(JsonObject.class);
assertThat(dedicatedVehicle.getString("brand"),containsString("Opel"));
assertThat(dedicatedVehicle.getString("type"),equalTo("Commodore"));
```

DELETE einer REST-Ressource

Test-Client

delete() - Methode

VehicleResourceIT.java

```
61        assertThat(response.getStatus(),is(200));
62        JsonArray allTodos = response.readEntity(JsonArray.class);
63        System.out.println("payload = " + allTodos);
64        assertThat(allTodos,not(empty()));
65
66        JsonObject vehicle = allTodos.getJsonObject(0);
67        assertThat(vehicle.getString("brand"),equalTo("Opel"));
68        assertThat(vehicle.getString("type"), startsWith("Commodore"));
69
70        Response deleteResponse = this.target
71                .path("42")
72                .request(MediaType.APPLICATION_JSON)
73                .delete();
74        assertThat(deleteResponse.getStatus(),is(204)); // no content
75    }
```

1 test failed - 1s 411ms

```
411ms
411ms   Actual   :<500>
411ms    <Click to see difference>

      at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
      at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:8)
      at at.htl.vehicle.rest.VehicleResourceIT.crud(VehicleResourceIT.java:74)
    <35 internal calls>

 Process finished with exit code 255
```

Jetzt schlägt nur noch ein Test fehl. Eine Resource soll gelöscht werden, die nicht exisitiert.

Es muss daher die Methode delete() geändert werden

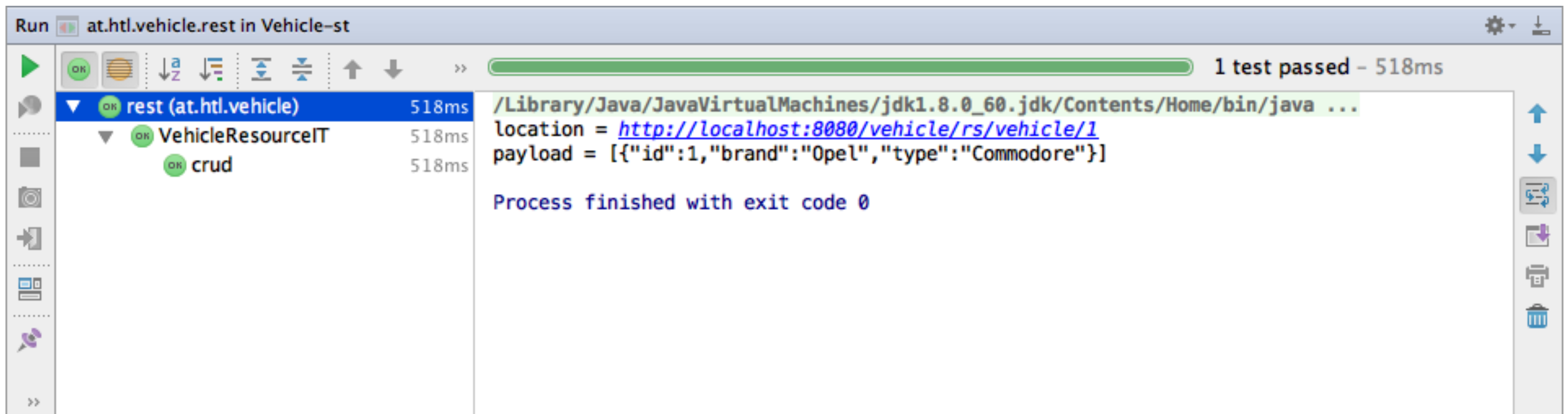
delete()

```java
public void delete(long id) {
    try {
        Vehicle reference = this.em.getReference(Vehicle.class, id);
        this.em.remove(reference);
    } catch (EntityNotFoundException e) {
        // it's already removed ...
    }
}
```

- Der Zugriff auf die DB muss mit einer EntityNotFoundException abgefangen werden.
- Die Exception wird hier ignoriert, da eine Entity mit dieser Id sowieso nicht in der DB existiert.
- Man könnte diese Exception auch eskalieren und ggf. ein Rollback durchführen (als Teil eines Geschäftsprozesses)

Test-Client

Die Tests funktionieren



Test crud()

Test-Client

```
@Test
public void crud() {
    JsonObjectBuilder vehicleBuilder = Json.createObjectBuilder();
    JsonObject vehicleToCreate = vehicleBuilder
            .add("brand", "Opel")
            .add("type", "Commodore")
            .build();

    // create
    Response postResponse = this.target
            .request().post(Entity.json(vehicleToCreate));
    assertThat(postResponse.getStatus(),is(201));
    String location = postResponse.getHeaderString("Location");
    System.out.println("location = " + location);

    // find
    JsonObject dedicatedVehicle = this.client
            .target(location)
            .request(MediaType.APPLICATION_JSON)
            .get(JsonObject.class);
    assertThat(dedicatedVehicle.getString("brand"),containsString("Opel"));
    assertThat(dedicatedVehicle.getString("type"),equalTo("Commodore"));

    // findAll
    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"));

    // deleting not-existing
    Response deleteResponse = this.target
            .path("42")
            .request(MediaType.APPLICATION_JSON)
            .delete();
    assertThat(deleteResponse.getStatus(),is(204)); // no content
}
```

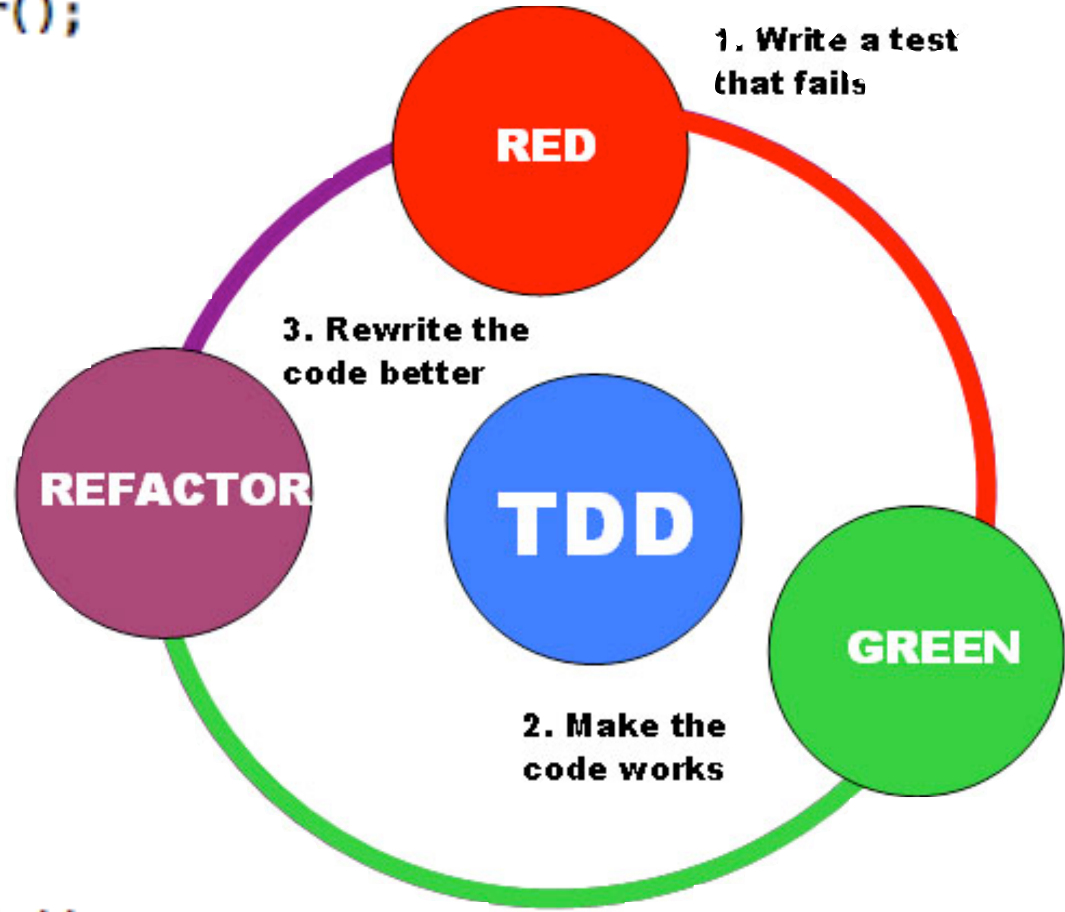
Das Problem hier ist, dass nicht zuviel Funktionalität in einem Testfall enthalten sein soll. Weiters sollen die einzelnen Testfälle voneinander unabhängig sein.

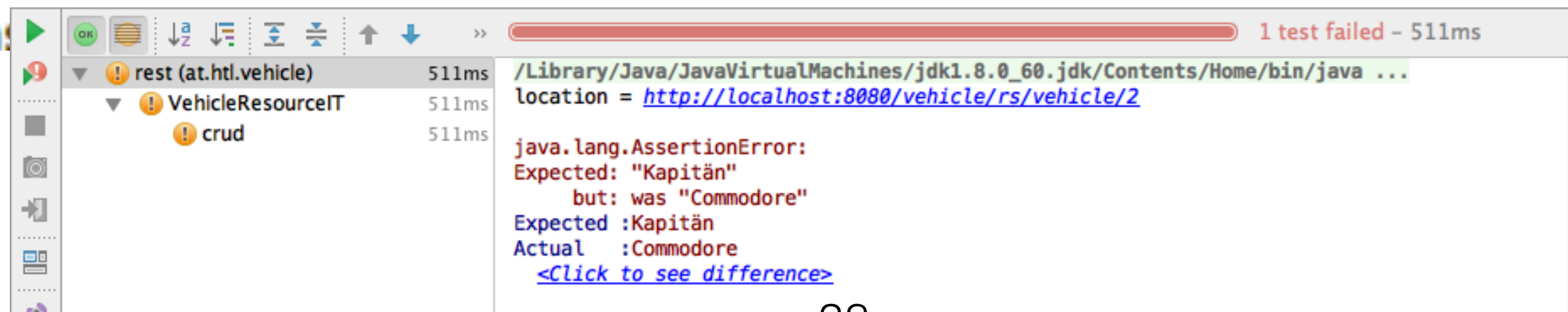
UPDATE der REST-Ressource

Test-Client

Erstellen des Tests

```
// update
JsonObjectBuilder updateBuilder = Json.createObjectBuilder();
JsonObject updated = updateBuilder
        .add("brand", "Opel")
        .add("type", "Kapitän")
        .build();

this.client
        .target(location)
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(updated));

// find it again
JsonObject updatedVehicle = this.client
        .target(location)
        .request(MediaType.APPLICATION_JSON)
        .get(JsonObject.class);
assertThat(updatedVehicle.getString("brand"),equalTo("Opel"));
assertThat(updatedVehicle.getString("type"),equalTo("Kapitän"));

// findAll
Response respon
```





TDD

ALL CODE IS GUILTY UNTIL PROVEN INNOCENT

Methode update()

```
@PUT
@Path("{id}")
public Vehicle update(@PathParam("id") long id, Vehicle vehicle) {
    vehicle.setId(id);
    return vehicleFacade.save(vehicle);
}
```

- Methode update() kann die Methode save(…) verwenden, da diese wiederum merge(…) verwendet. merge() funktioniert wie ein UPSERT (INSERT oder UPDATE)
- Für die Id wird die vom Pfad mitgegebene Id verwendet (und nicht die Id vom Objekt)

Test-Client

Testergebnis

Run at.htl.vehicle.rest in Vehicle-st

1 test passed – 548ms

- rest (at.htl.vehicle) 548ms
  - VehicleResourceIT 548ms
    - crud 548ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
location = http://localhost:8080/vehicle/rs/vehicle/1
payload = [{"id":1,"brand":"Opel","type":"Kapitän"}]

Process finished with exit code 0
```

Implementing a boolean Property

Vehicle.java

```java
@Entity
@NamedQuery(name = "Vehicle.findAll", query = "SELECT v FROM Vehicle v")
@XmlRootElement
public class Vehicle {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String brand;
    private String type;
    private boolean annualVignetteValid;

    public Vehicle() {
    }

    public Vehicle(String brand, String type) {
```

VehicleFacade.java

```java
public Vehicle updateVignetteValid(long id, boolean vignetteValid) {
    Vehicle vehicle = this.findById(id);
    vehicle.setAnnualVignetteValid(vignetteValid);
    return vehicle;
}
```

VehicleResource.java

```
@PUT
@Path("{id}/vignette_valid")
public Vehicle update(@PathParam("id") long id, JsonObject vignetteValidUpdate) {
    boolean vignetteValid = vignetteValidUpdate.getBoolean("annualVignetteValid");
    return vehicleFacade.updateVignetteValid(id, vignetteValid);
}
```

Test-Client

VehicleResourceIT.java

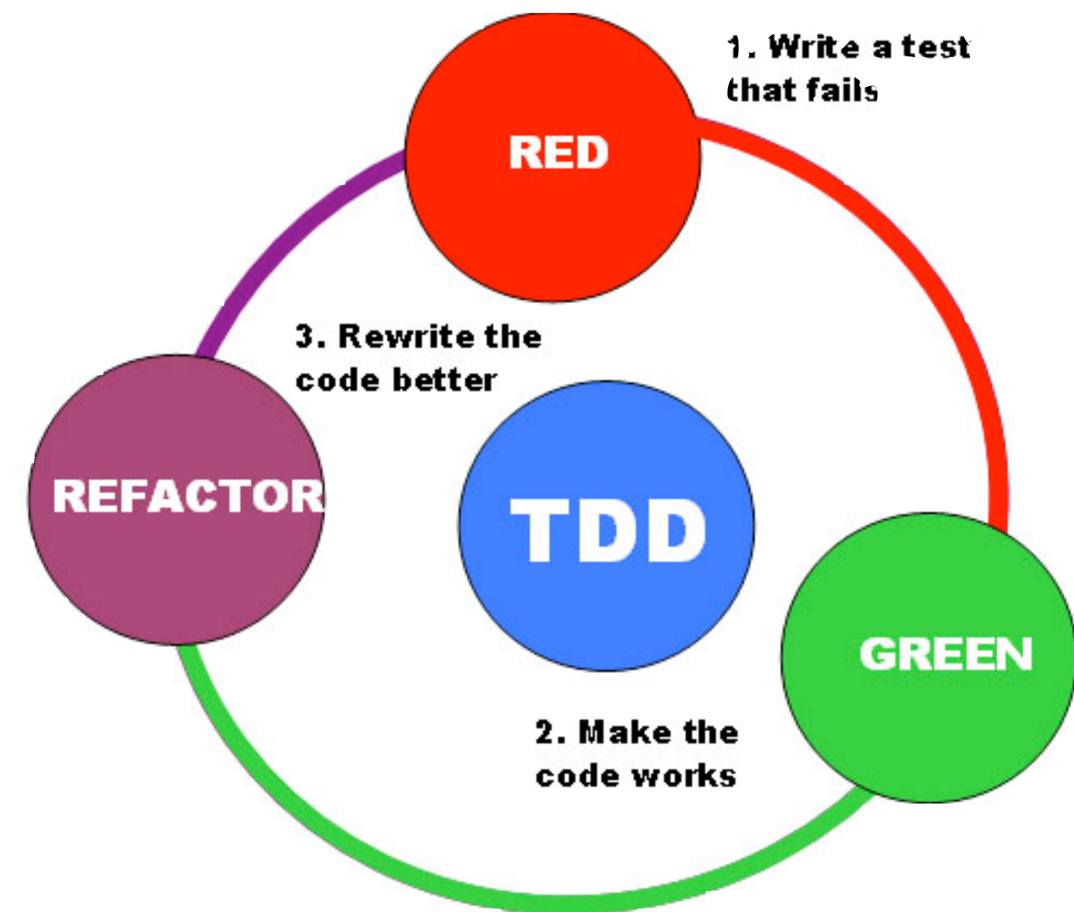
```java
// update vignetteValid
JsonObjectBuilder vignetteValidBuilder = Json.createObjectBuilder();
JsonObject vignetteValid = updateBuilder
        .add("annualVignetteValid", true)
        .build();

this.client
        .target(location)
        .path("vignette_valid")
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(vignetteValid));

// verify vignetteValid
updatedVehicle = this.client
        .target(location)
        .request(MediaType.APPLICATION_JSON)
        .get(JsonObject.class);
assertThat(updatedVehicle.getBoolean("annualVignetteValid"),equalTo(true));
```

Wie sieht es nun aus?

- Die boolean-Variable annualVignetteValid wurde implementiert und als Resource zur Verfügung gestellt.
- Allerdings wurde bislang die Probleme nicht beachtet:
    - In VehicleFacade kann in der Methode updateVignetteValid() der boolean-Parameter null sein
    - In VehicleResource kann in der Methode updateVignetteValid() der boolean-Parameter nicht verfügbar sein

Vorgangsweise

- Die Sonderfälle in den Systemtests abbilden
- Anschließend die Sonderfälle in der Facade und der Resource behandeln.



Test-Client

Der neue Test

Ein gültiges Json-Objekt wird an ein ungültiges Vehicle gesendet

```
// update not existing  vignetteValid
JsonObjectBuilder notExistingBuilder = Json.createObjectBuilder();
vignetteValid = notExistingBuilder
        .add("annualVignetteValid", true)
        .build();

Response response = this.target
        .path("-42")
        .path("vignette_valid")
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(vignetteValid));
assertThat(response.getStatus(),is(400));
assertThat(response.getHeaderString("reason"), not(isEmptyString()));
```

Eine gültige vignetteValid wird an eine nicht existierende Id gesendet

Test-Client

Der erste Testlauf

```
37      public Vehicle save(Vehicle vehicle) { return this.em.merge(vehicle); }
40
41      public Vehicle updateVignetteValid(long id, boolean vignetteValid) {
42          Vehicle vehicle = this.findById(id);
43          vehicle.setAnnualVignetteValid(vignetteValid);
44          return vehicle;
45      }
46  }
47
48
49
```

Output

```
    at org.jboss.resteasy.core.ResourceMethodInvoker.invoke(ResourceMethodInvoker.java:237)
    at org.jboss.resteasy.core.SynchronousDispatcher.invoke(SynchronousDispatcher.java:356)
    ... 32 more
Caused by: java.lang.NullPointerException
    at at.htl.vehicle.business.VehicleFacade.updateVignetteValid(VehicleFacade.java:43) <4 internal calls>
    at org.jboss.as.ee.component.ManagedReferenceMethodInterceptor.processInvocation
  (ManagedReferenceMethodInterceptor.java:52)
    at org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:340)
    at org.jboss.invocation.InterceptorContext$Invocation.proceed(InterceptorContext.java:437)
    at org.jboss.as.weld.ejb.Jsr299BindingsInterceptor.doMethodInterception(Jsr299BindingsInterceptor.java:82)
    at org.jboss.as.weld.ejb.Jsr299BindingsInterceptor.processInvocation(Jsr299BindingsInterceptor.java:93)
```

Änderungen im Code

VehicleFacade.java

```java
41    public Vehicle updateVignetteValid(long id, boolean vignetteValid) {
42        Vehicle vehicle = this.findById(id);
43        if (vehicle == null) {
44            return null;
45        }
46        vehicle.setAnnualVignetteValid(vignetteValid);
47        return vehicle;
48    }
```

VehicleResource.java

```java
49    @PUT
50    @Path("{id}/vignette_valid")
51    public Response updateVignetteValid (@PathParam("id") long id, JsonObject vignetteValidUpdate) {
52        boolean vignetteValid = vignetteValidUpdate.getBoolean("annualVignetteValid");
53        Vehicle vehicle = vehicleFacade.updateVignetteValid(id, vignetteValid);
54        if (vehicle == null) {
55            return Response
56                    .status(Response.Status.BAD_REQUEST)
57                    .header("reason", "vehicle with id " + id + " does not exist")
58                    .build();
59        } else {
60            return Response.ok(vehicle).build();
61        }
62    }
```

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.2

Test-Client

Der neuer Test

Ein <u>ung</u>ültiges Json-Objekt wird an ein gültiges Vehicle gesendet

```
// update malformed vignetteValid
notExistingBuilder = Json.createObjectBuilder();
vignetteValid = notExistingBuilder
        .add("something wrong", true)
        .build();

response = this.client
        .target(location)
        .path("vignette_valid")
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(vignetteValid));
assertThat(response.getStatus(),is(400));
assertThat(response.getHeaderString("reason"), not(isEmptyString()));
```

Test-Client

Der erste Testlauf

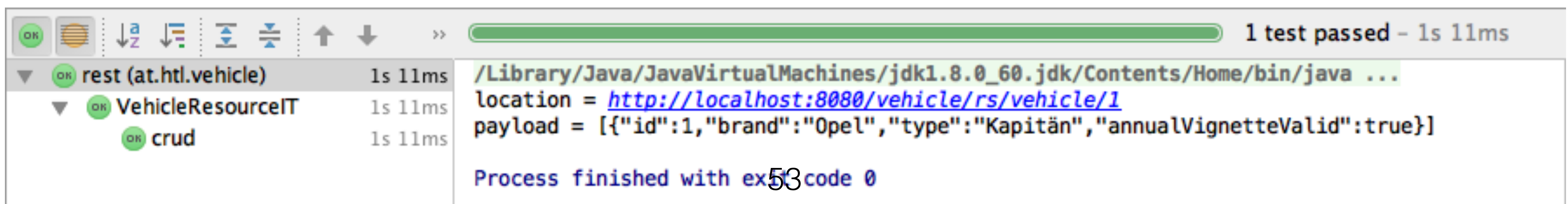
```
54      @PUT
55      @Path("{id}/vignette_valid")
56      public Response updateVignetteValid (@PathParam("id") long id, JsonObject vignetteValidUpdate) {
57          boolean vignetteValid = vignetteValidUpdate.getBoolean("annualVignetteValid");
58          Vehicle vehicle = vehicleFacade.updateVignetteValid(id, vignetteValid);
59          if (vehicle == null) {
60              return Response
61                      .status(Response.Status.BAD_REQUEST)
62                      .header("reason", "vehicle with id " + id + " does not exist")
63                      .build();
64          } else {
65              return Response.ok(vehicle).build();
66          }
67      }
68
```

Output

```
    at org.jboss.resteasy.core.ResourceMethodInvoker.invoke(ResourceMethodInvoker.java:250)
    at org.jboss.resteasy.core.ResourceMethodInvoker.invoke(ResourceMethodInvoker.java:237)
    at org.jboss.resteasy.core.SynchronousDispatcher.invoke(SynchronousDispatcher.java:356)
    ... 32 more
Caused by: java.lang.NullPointerException
    at org.glassfish.json.JsonObjectBuilderImpl$JsonObjectImpl.getBoolean(JsonObjectBuilderImpl.java:226)
    at at.htl.vehicle.rest.VehicleResource.updateVignetteValid(VehicleResource.java:52) <4 internal calls>
    at org.jboss.as.ee.component.ManagedReferenceMethodInterceptor.processInvocation
 (ManagedReferenceMethodInterceptor.java:52)
    at org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:340)
    at org.jboss.invocation.InterceptorContext$Invocation.proceed(InterceptorContext.java:437)
```

Änderungen im Code

```
@PUT
@Path("{id}/vignette_valid")
public Response updateVignetteValid (@PathParam("id") long id, JsonObject vignetteValidUpdate) {
    if (!vignetteValidUpdate.containsKey("annualVignetteValid")) {
        return Response.status(Response.Status.BAD_REQUEST)
                .header("reason", "JSON should contain field annualVignetteValid")
                .build();
    }
    boolean vignetteValid = vignetteValidUpdate.getBoolean("annualVignetteValid");
    Vehicle vehicle = vehicleFacade.updateVignetteValid(id, vignetteValid);
    if (vehicle == null) {
        return Response
                .status(Response.Status.BAD_REQUEST)
                .header("reason", "vehicle with id " + id + " does not exist")
                .build();
    } else {
        return Response.ok(vehicle).build();
    }
}
```



Sub Resources

REST Sub Resource

- Sub Resources werden verwendet um große, unübersichtliche REST-Resourcen zu vereinfachen und übersichtlicher zu gestalten.

- Dabei werden bestimmte Requests an eine andere Klasse weitergeleitet.

- In unserem Beispiel wird zunächst der Name der REST-Resource von VehicleResource auf VehiclesResource geändert. (Eigentlich hätte diese Klasse von Beginn an so heißen sollen)

Umbenannte Klasse

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

import ...

@Stateless
@Path("vehicle")
public class VehiclesResource {

    @Inject
    VehicleFacade vehicleFacade;

    @GET
    @Path("{id}")
    public Vehicle find(@PathParam("id") long id) { return vehicleFacade.findById(id); }

    @GET
    public List<Vehicle> findAll() { return vehicleFacade.findAll(); }

    @DELETE
    @Path("{id}")
    public void delete(@PathParam("id") long id) { vehicleFacade.delete(id); }

    @PUT
    @Path("{id}")
    public Vehicle update(@PathParam("id") long id, Vehicle vehicle) {...}

    @PUT
    @Path("{id}/vignette_valid")
    public Response updateVignetteValid (@PathParam("id") long id, JsonObject vignetteValidUpdate) {...}

    @POST
    public Response save(Vehicle vehicle, @Context UriInfo info) {...}
}
```

REST Sub Resource

- Resource classes are able to partially process a request and provide another "sub" resource object that can process the remainder of the request.
- **Resource methods that have a @Path annotation, but no HTTP method are considered sub-resource locators**. Their job is to provide an object that can process the request.
- The getCustomer() method is a sub-resource locator method.

```
@Path("/")
public class ShoppingStore {

   @Path("/customers/{id}")
   public Customer getCustomer(@PathParam("id") int id) {
      Customer cust = ...; // Find a customer object
      return cust;
   }
}
```

```
public class Customer {

    @GET
    public String get() {...}

    @Path("/address")
    public String getAddress() {...}

}
```

https://docs.jboss.org/resteasy/docs/1.0.2.GA/userguide/html/JAX-RS_Resource_Locators_and_Sub_Resources.html

Vorgangsweise

- Es werden alle Methoden, die eine {id} im Pfad haben, in eine eigene Klasse VehicleResource ausgelagert
- Jede dieser Methoden beschäftigt sich nur mit je einem Vehicle
- Die Id für dieses Vehicle wird im Aufruf der Klasse übergeben. Daher ändern sich die Methoden-Signaturen

VehiclesResources.java

```java
@DELETE
@Path("{id}")
public void delete(@PathParam("id") long id) {
    vehicleFacade.delete(id);
}
```

VehicleResources.java

```java
@DELETE
public void delete() {
    vehicleFacade.delete(id);
}
```

VehicleResource.java

```java
public class VehicleResource {

    long id;
    VehicleFacade vehicleFacade;

    public VehicleResource(long id, VehicleFacade vehicleFacade) {...}

    @PUT
    public Vehicle update(Vehicle vehicle) {...}

    @GET
    public Vehicle find() { return vehicleFacade.findById(id); }

    @DELETE
    public void delete() { vehicleFacade.delete(id); }

    @PUT
    @Path("/vignette_valid")
    public Response updateVignetteValid (JsonObject vignetteValidUpdate) {...}
}
```

Nur {id} wird als Parameter entfernt

Nur {id} wird vom Pfad entfernt

VehiclesResource

Nur die Methoden, die mehr als ein Vehicle betreffen (findAll) oder keine {id} in der Parameterleiste besitzen (save), sind enthalten

```
@Stateless
@Path("vehicle")
public class VehiclesResource {

    @Inject
    VehicleFacade vehicleFacade;

    @Path("{id}")
    public VehicleResource find(@PathParam("id") long id) {
        return new VehicleResource(id, vehicleFacade);
    }

    @GET
    public List<Vehicle> findAll() { return vehicleFacade.findAll(); }


    @POST
    public Response save(Vehicle vehicle, @Context UriInfo info) {...}
}
```

Diese Methode gibt den Kontrollfluss an die Sub Resource weiter

Test-Client

Testergebnis

Run at.htl.vehicle.rest in Vehicle-st

1 test passed – 1s 32ms

Test	Time
rest (at.htl.vehicle)	1s 32ms
VehicleResourceIT	1s 32ms
crud	1s 32ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
location = http://localhost:8080/vehicle/rs/vehicle/1
payload = [{"id":1,"brand":"Opel","type":"Kapitän","annualVignetteValid":true}]

Process finished with exit code 0
```

- Die Tests funktionieren unverändert
- Die REST Schnittstelle ist nun klarer strukturiert und übersichtlicher

Optimistic Locking

Problematik

- Auf unseren REST-Service können nun verschiedenste Geräte (User) zugreifen. Man spricht hierbei von Concurrency (Gleichzeitigkeit)

- Werden nun Daten von verschiedenen Usern verändert, kann es passieren, dass durch Caching Daten nicht sofort aktualisert werden und diese Änderungen verloren gehen (vgl. Probleme bei Concurrency)

- Daher ist ein (optimistisches) Locking sinnvoll

Optimistic Locking

```java
@Entity
@NamedQuery(name = "Vehicle.findAll", query = "SELECT v
@XmlRootElement
public class Vehicle {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String brand;
    private String type;
    private boolean annualVignetteValid;

    @Version
    private long version;

    public Vehicle() {
    }

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

    Getter and setter
}
```

- Eine Variable wird mit @Version annotiert (Number Datentyp)
- Die Bezeichnung dieser Variable ist frei
- JPA inkrementiert diese Variable bei jeder Datenänderung
- Bei zu ändernden Daten, wird deren Versionsnr mit der aktuellen Versionsnr in der DB verglichen.
- Stimmt diese Versionsnr nicht überein, wird ein OptimisticLockError geworfen

Test-Client

Testfälle

- Zum Testen des Optimistic Lockings sind zumindest zwei Updates notwendig

```
// update
JsonObjectBuilder updateBuilder = Json.createObjectBuilder();
JsonObject updated = updateBuilder
        .add("brand", "Opel")
        .add("type", "Kapitän")
        .build();

Response updateResonse = this.client
        .target(location)
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(updated));
assertThat(updateResonse.getStatus(), is(200));

// update again
updateBuilder = Json.createObjectBuilder();
updated = updateBuilder
        .add("brand", "VW")
        .add("type", "Käfer 1400")
        .build();

updateResonse = this.client
        .target(location)
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(updated));
assertThat(updateResonse.getStatus(), is(200));
```

Vehicle mit Id = 1

Vehicle mit Id = 1

Test-Client

Testergebnis

ClientLog

1 test failed – 1s 152ms

rest (at.htl.vehicle) 1s 152ms
VehicleResourceIT 1s 152ms
crud 1s 152ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
location = http://localhost:8080/vehicle/rs/vehicle/1

java.lang.AssertionError:
Expected: is <200>
     but: was <500>
Expected :is <200>

Actual   :<500>
  <Click to see difference>
```

ServerLog

```
    at org.jboss.as.ejb3.tx.CMTTxInterceptor.invokeInOurT
    ... 86 more
Caused by: org.hibernate.StaleObjectStateException: Row was updated or deleted by another transaction (or
 unsaved-value mapping was incorrect) : [at.htl.vehicle.entity.Vehicle#1] <7 internal calls>
    ... 117 more

10:16:11,181 ERROR [io.undertow.request] (default task-7) UT005023: Exception handling request to
 /vehicle/rs/vehicle/1: org.jboss.resteasy.spi.UnhandledException: javax.ejb.EJBException: javax.persistence
 .OptimisticLockException: Row was updated or deleted by another transaction (or unsaved-value mapping was
 incorrect) : [at.htl.vehicle.entity.Vehicle#1]
    at org.jboss.resteasy.core.ExceptionHandler.handleApplicationException(ExceptionHandler.java:76)
```

Die Hibernate-Exception StaleObjectStateException ist in einer JPA-Exception gewrapped: OptimisticLockException

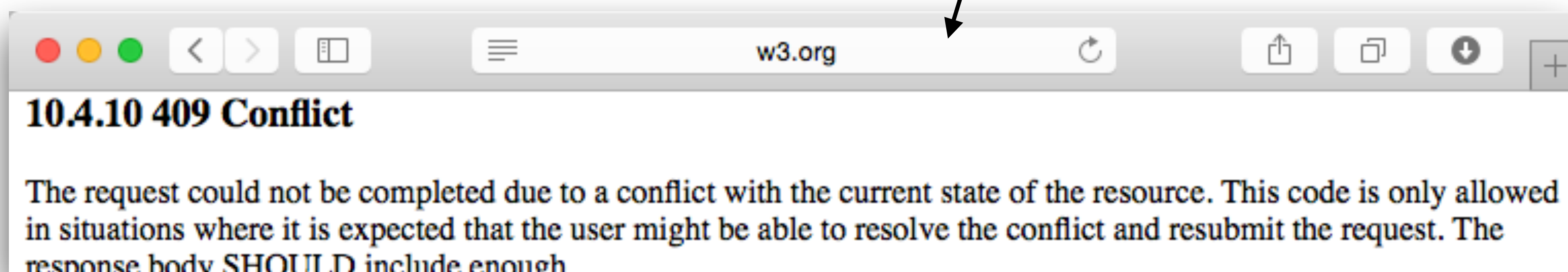
Abhilfe

- Man könnte nun in der VehicleFacade alle Methoden, die in Frage kommen, mit try-catch Blöcken versehen, die eine OptimisticLockException abfangen, oder man verwendet
- ExceptionMapper

ExceptionMapper

EJBEcxeptionMapper.java

```java
@Provider
public class EJBExceptionMapper implements ExceptionMapper<EJBException> {
    @Override
    public Response toResponse(EJBException ex) {
        Throwable cause = ex.getCause();
        Response unkownError = Response.serverError()
                .header("cause", ex.toString())
                .build();
        if (cause instanceof OptimisticLockException) {
            return Response.status(Response.Status.CONFLICT)
                    .header("cause", "conflict occured " + cause)
                    .build();
        }
        return unkownError;
    }
}
```


10.4.10 409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body SHOULD include enough

Test-Client

Testergebnis

Run at.htl.vehicle.rest in Vehicle-st

1 test failed – 999ms

- rest (at.htl.vehicle) 999ms
  - VehicleResourceIT 999ms
    - crud 999ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
location = http://localhost:8080/vehicle/rs/vehicle/1

java.lang.AssertionError:
Expected: is <200>
     but: was <409>
Expected :is <200>

Actual   :<409>
 <Click to see difference>
```

- Test muss noch korrigiert werden
- Es wird der korrekte Wert 409 zurückgegeben, aber 200 erwartet
-

Test-Client

Aktualisierter Test

```java
// update again
updateBuilder = Json.createObjectBuilder();
updated = updateBuilder
        .add("brand", "VW")
        .add("type", "Käfer 1400")
        .build();

updateResponse = this.client
        .target(location)
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(updated));
assertThat(updateResponse.getStatus(), is(409));
String conflictInformation = updateResponse.getHeaderString("cause");
assertThat(conflictInformation, not(isEmptyString()));
System.out.println("conflictInformation = " + conflictInformation);
```

Run at.htl.vehicle.rest in Vehicle-st

1 test passed – 1s 25ms

Test	Time
rest (at.htl.vehicle)	1s 25ms
VehicleResourceIT	1s 25ms
crud	1s 25ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
location = http://localhost:8080/vehicle/rs/vehicle/1
conflictInformation = conflict occured javax.persistence.OptimisticLockException:
 Row was updated or deleted by another transaction (or unsaved-value mapping was
 incorrect) : [at.htl.vehicle.entity.Vehicle#1]
payload = [{"id":1,"brand":"Opel","type":"Kapitän","annualVignetteValid":true}]

Process finished with exit code 0
```

JAX-RS and Bean Validation

Problematik

- Derzeit akzeptiert unsere REST-Resource alle Werte
- Die Marke (brand) oder der Typ (type) eines Fahrzeugs könnten leer sein. Diese Fahrzeuginformationen wären sinnlos.
- Daher sind die Daten vor dem persistieren zu validieren (auf Gültigkeit überprüfen).
- In JEE gibt es dazu die Bean Validation

Erste Constraints ...

```
@Entity
@NamedQuery(name = "Vehicle.findAll", query = "SELECT v FROM Vehicle v")
@XmlRootElement
public class Vehicle {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    @Size(min = 1, max = 256)
    private String brand;
    private String type;
    private boolean annualVignetteValid;
```

Test-Client

Testergebnis

Run at.htl.vehicle.rest in Vehicle-st

2 tests done: 1 failed

- rest (at.htl.vehicle) 1s 102ms
  - VehicleResourceIT 1s 102ms
    - crud 1s 53ms
    - createVehicleWithout 49ms

```
java.lang.AssertionError:
Expected: is <400>
     but: was <500>
Expected :is <400>

Actual   :<500>
  <Click to see difference>
```

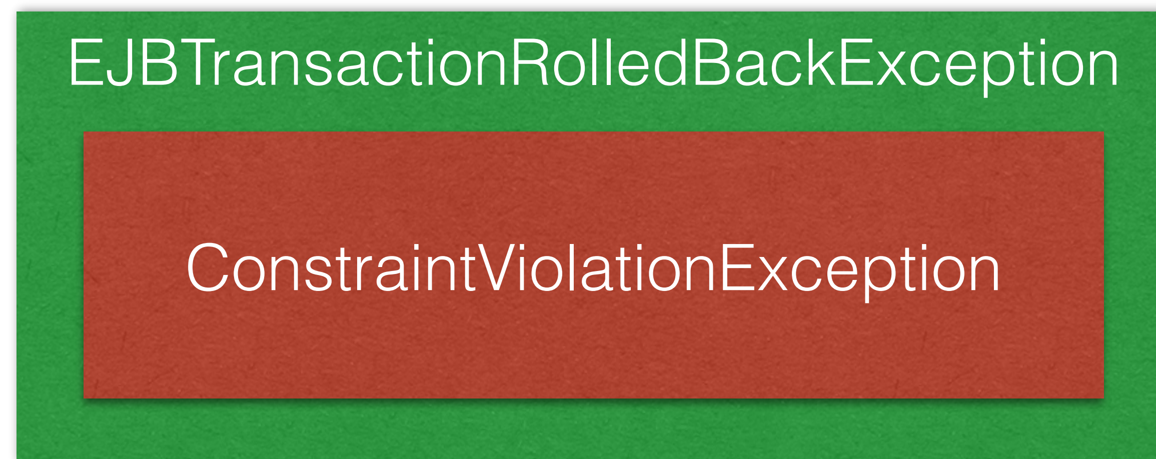
Output

```
11:27:34,596 ERROR [org.jboss.as.ejb3.invocation] (default task-15) WFLYEJB0034: EJB Invocation failed on
component VehicleFacade for method public at.htl.vehicle.entity.Vehicle at.htl.vehicle.business.VehicleFacade
.save(at.htl.vehicle.entity.Vehicle): javax.ejb.EJBTransactionRolledbackException: Validation failed for
classes [at.htl.vehicle.entity.Vehicle] during persist time for groups [javax.validation.groups.Default, ]
List of constraint violations:[
   ConstraintViolationImpl{interpolatedMessage='may not be null', propertyPath=brand, rootBeanClass=class
 at.htl.vehicle.entity.Vehicle, messageTemplate='{javax.validation.constraints.NotNull.message}'}
]
   at org.jboss.as.ejb3.tx.CMTTxInterceptor.handleInCallerTx(CMTTxInterceptor.java:159)
   at org.jboss.as.ejb3.tx.CMTTxInterceptor.invokeInCallerTx(CMTTxInterceptor.java:256)
```

```
Caused by: javax.validation.ConstraintViolationException: Validation failed for classes [at.htl.vehicle.entity
.Vehicle] during persist time for groups [javax.validation.groups.Default, ]
```

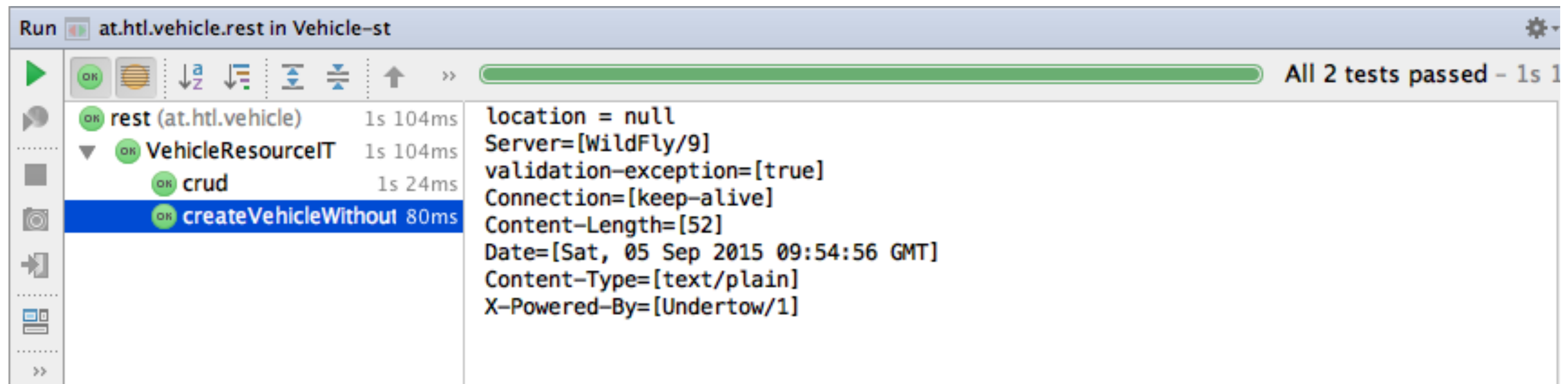
Eine ConstraintViolationException ist in einer EJBTransactionRolledBackException gewrapped

Exceptions



- Will man nun die ConstraintViolationException abfangen (ErrorCode 400), müsste man die Exception rekursiv extrahieren
- Der Wert von „brand“ wird erst beim Flush in die Datenbank überprüft, daher wird auch die EJBTransactionRolledBackException geworfen.
- Überprüft man den Wert von „brand“ bereits bei der Übergabe an die REST-Resource, könnte man sich das Rollback ersparen
- Durch Verwendung der Annotation @Valid kann der Zeitpunkt der Validierung dementsprechend geändert werden

@Valid



- Nun gelingt der Test
- Im Header sieht man an validation-exception=[true] dass eine Validation Exception geworfen wurde
- Die Location ist null, da kein Datensatz persistiert wurde

Funktionierender Test

```java
@Test
public void createValidVehicle() {
    JsonObjectBuilder vehicleBuilder = Json.createObjectBuilder();
    JsonObject vehicleToCreate = vehicleBuilder
            .add("brand", "VW")
            .add("type", "Käfer 1400")
            .build();

    Response postResponse = this.target
            .request()
            .post(Entity.json(vehicleToCreate));
    assertThat(postResponse.getStatus(),is(201));
    String location = postResponse.getHeaderString("Location");
    System.out.println("location = " + location);
}
```

Run at.htl.vehicle.rest in Vehicle-st

All 3 tests passed – 1s 306r

- rest (at.htl.vehicle) 1s 306ms
  - VehicleResourceIT 1s 306ms
    - crud 1s 174ms
    - createValidVehicle 35ms
    - createVehicleWithout 97ms

```
location = http://localhost:8080/vehicle/rs/vehicle/2
```

Datenbankinhalt

ID	ANNUALVIGNETTEVALID	BRAND	TYPE	VERSION
1	☑	Opel	Kapitän	2
2	☐	VW	Käfer 1400	0

https://books.google.at/books?id=MqjKLdZCnZIC&lpg=PP1&dq=gonzalves%20java%20ee%207&hl=de&pg=PA74#v=onepage&q=gonzalves%20java%20ee%207&f=false

Built-In Constraints

Constraint	Accepted Types	Description
AssertFalse AssertTrue	Boolean, boolean	The annotated element must be either false or true
DecimalMax DecimalMin	BigDecimal, BigInteger, CharSequence, byte, short, int, long, and respective wrappers	The element must be greater or lower than the specified value
Future Past	Calendar, Date	The annotated element must be a date in the future or in the past
Max Min	BigDecimal, BigInteger, byte, short, int, long, and their wrappers	The element must be greater or lower than the specified value
Null NotNull	Object	The annotated element must be null or not
Pattern	CharSequence	The element must match the specified regular expression
Digits	BigDecimal, BigInteger, CharSequence, byte, short, int, long, and respective wrappers	The annotated element must be a number within accepted range
Size	Object[], CharSequence, Collection<?>, Map<?, ?>	The element size must be between the specified boundaries

Cross Field Validation

Cross Field Validation

- In der Realität sind die Constraints oft voneinander abhängig.
- Man spricht hierbei von Cross Field Validation

Vorbereitungen

- Am Server in pom.xml eine Dependency für junit eintragen:

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
```

Ziel

- Es wird ein Validator erstellt, der sicherstellt, dass wenn die Marke (brand)mindestens 10 Zeichen lang ist, auch der Typ (type) mindestens 10 Zeichen lang ist.

- Hierzu wird zunächst ein Interface erstellt

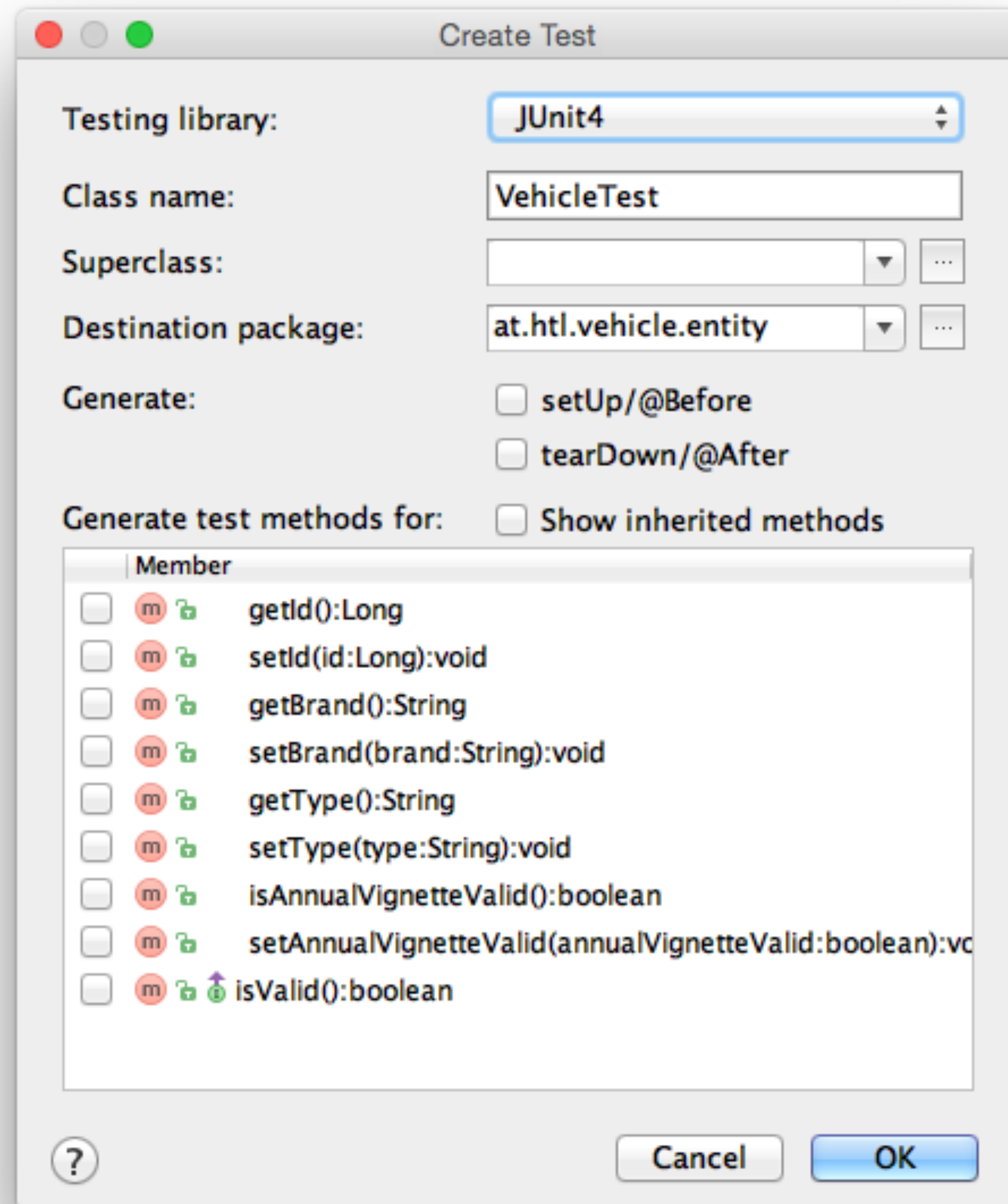
```
public interface ValidEntity {

    boolean isValid();

}
```

Vehicle.java

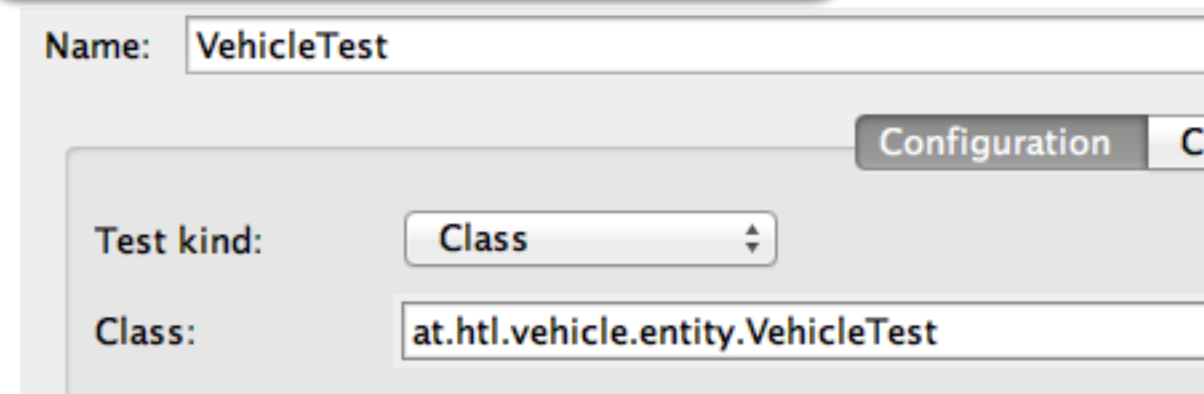
- Anschließend implementiert die gewünschte Entität das Interface

```java
@Entity
@NamedQuery(name = "Vehicle.findAll", query = "SELECT v FROM Vehicle v")
@XmlRootElement
public class Vehicle implements ValidEntity {

    ...

    @Override
    public boolean isValid() {
        if (this.brand.length() > 10) {
            return this.type.length() > 10;
        }
        return true;
    }
    //endregion
}
```

Testfall am Server erstellen



Run/Debug konfigurieren



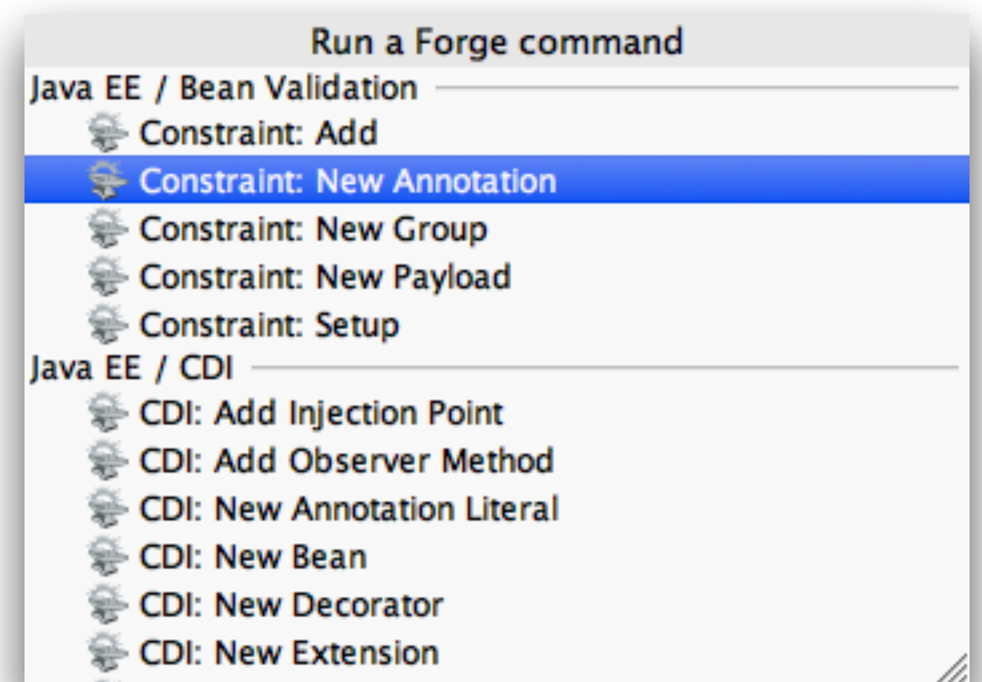
Frage

- Was sind die Unterschiede zwischen Testfällen am Server und am (Jersey-)Client

Custom Validator

- Damit die Cross Field Validation auch in der REST-Resource verwendet werden kann, ist ein Custom Validator zu erstellen (auch Generic Constraint, vgl Gonzalves S.76)

- Ein Custom Validator besteht aus einer
  - Annotation und einem
  - ConstraintValidator



https://books.google.at/books?id=Rr1OwYRj8XMC&lpg=PP1&hl=de&pg=PP1#v=onepage&q&f=false

Custom Validator

Validator

```java
public class CrossCheckConstraintValidator
        implements ConstraintValidator<CrossCheck, ValidEntity> {
    @Override
    public void initialize(CrossCheck constraintAnnotation) {

    }

    @Override
    public boolean isValid(ValidEntity entity,
                           ConstraintValidatorContext context) {
        return entity.isValid();
    }
}
```

Annotation

```java
@Documented
@Constraint(validatedBy = CrossCheckConstraintValidator.class)
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface CrossCheck {

    String message() default "Cross check failed!";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}
```

```java
@Entity
@NamedQuery(name = "Vehicle.findAll", query = "SELECT v FROM Vehicle v")
@XmlRootElement
@CrossCheck
public class Vehicle implements ValidEntity {

    ...

    @Override
    public boolean isValid() {
        if (this.brand.length() > 10) {
            return this.type.length() > 10;
        }
        return true;
    }
    //endregion
}
```

Anwendung der Constaint in Vehicle

Interface

```java
public interface ValidEntity {
    boolean isValid();
}
```

Test-Client

Testfall: Valides Vehicle

```java
@Test
public void createValidVehicle() {
    JsonObjectBuilder vehicleBuilder = Json.createObjectBuilder();
    JsonObject vehicleToCreate = vehicleBuilder
            .add("brand", "VW")
            .add("type", "Käfer 1400")
            .build();

    Response postResponse = this.target
            .request()
            .post(Entity.json(vehicleToCreate));
    postResponse.getHeaders().entrySet().forEach(System.out::println);
    assertThat(postResponse.getStatus(),is(201));
}
```

Run VehicleResourceIT.createValidVehicle

1 test passed – 1s 110ms

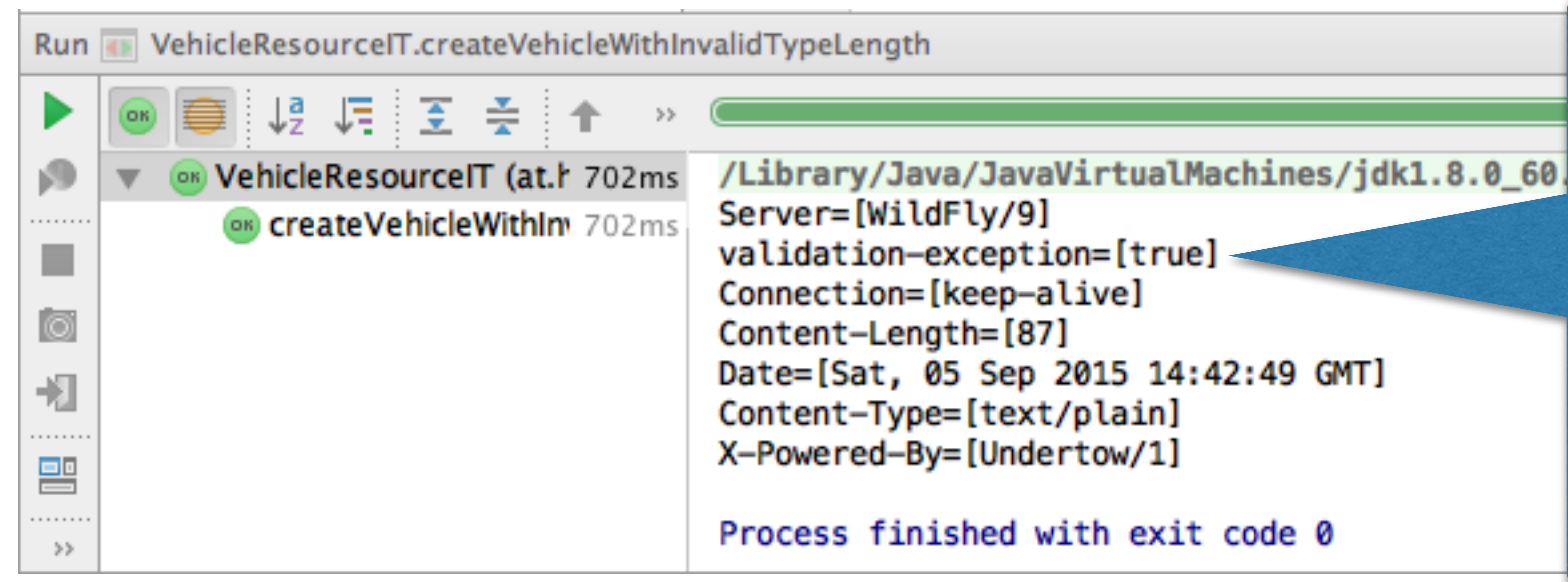
VehicleResourceIT (a 1s 110ms
createValidVehic 1s 110ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
Server=[WildFly/9]
Connection=[keep-alive]
Content-Length=[0]
Date=[Sat, 05 Sep 2015 14:39:07 GMT]
Location=[http://localhost:8080/vehicle/rs/vehicle/1]
X-Powered-By=[Undertow/1]

Process finished with exit code 0
```

Test-Client

Invalides Vehicle

```java
@Test
public void createVehicleWithInvalidTypeLength() {
    JsonObjectBuilder vehicleBuilder = Json.createObjectBuilder();
    JsonObject vehicleToCreate = vehicleBuilder
            .add("brand", "abcdefghijk")
            .add("type", "ABC")
            .build();

    Response postResponse = this.target
            .request()
            .post(Entity.json(vehicleToCreate));
    postResponse.getHeaders().entrySet().forEach(System.out::println);
    assertThat(postResponse.getStatus(),is(400));
}
```



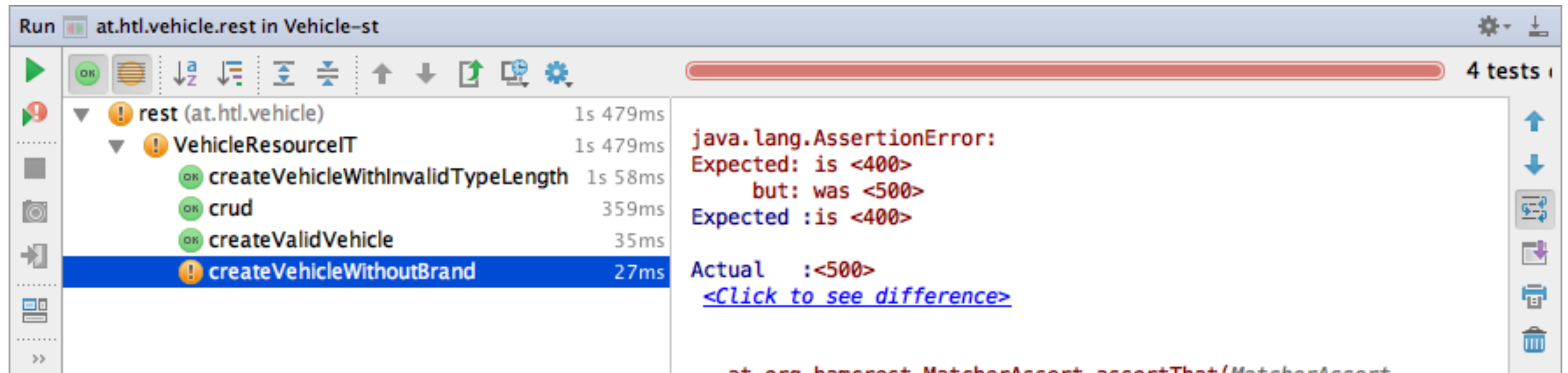
Das Insert schlägt mit Statuscode 400 fehl. Der Header - Eintrag validation-exception=[true]

Custom Validator

- Vorteil dieser Methode mit Interface: Man muss im Validator keine Validierungen implementieren, sondern dieser ruft nur die isValid()-Methoden in den Entities auf.
- Man muss kein Interface (hier: ValidEntity) erstellen und implementieren, sondern kann gleich im Validator zB auch gegen einen String validieren

```java
public class CheckCasevalidator {

    private CaseMode caseMode;

    public void initialize(CheckCase constraintAnnotation) {
        this.caseMode = constraintAnnotation.value();
    }

    public boolean isValid(String object, ConstraintValidatorContext constraintContext) {
        if (object == null)
            return true;
        if (caseMode == CaseMode.UPPER)
            return object.equals(object.toUpperCase());
        else
            return object.equals(object.toLowerCase());
    }
}
```

Problem



- Leider funktionieren nun nicht mehr alle Tests
- Es ist sehr wahrscheinlich, dass unsere Validierung dies verursacht.
- Daher sollte man einen geeigneten Unit-Test am Server erstellen

Test am Server

```java
@Test
public void vehicleWithoutBrand() {
    Vehicle valid = new Vehicle(null, "Kadett");
    assertThat(valid.isValid(),is(true));
}
```

```
1 test failed - 7ms
VehicleTest (at.htl.vehicle.entity)   7ms
  vehicleWithoutBrand                 7ms

java.lang.NullPointerException
	at at.htl.vehicle.entity.Vehicle.isValid(Vehicle.java:77)
	at at.htl.vehicle.entity.VehicleTest.vehicleWithoutBrand(VehicleTest.java:27)
  <26 internal calls>
```

```java
@Override
public boolean isValid() {
    if (brand == null) {
        return true;
    }
    if (this.brand.length() > 10) {
        return this.type.length() > 10;
    }
    return true;
}
```

In der Methode isValid() tritt schon ein Fehler auf (NPE weil brand == null) Dieser wird sowieso später abgefangen (@NotNull), daher wird er hier ignoriert.

```
1 test passed -
VehicleTest (at.htl.vehicle.entity)   5ms
  vehicleWithoutBrand                 5ms

/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Client Tests

Run at.htl.vehicle.rest in Vehicle-st

All 4 tes

Test	Time
rest (at.htl.vehicle)	1s 155ms
VehicleResourceIT	1s 155ms
createVehicleWithInvalidTypeLength	746ms
crud	358ms
createValidVehicle	28ms
createVehicleWithoutBrand	23ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bir
Server=[WildFly/9]
validation-exception=[true]
Connection=[keep-alive]
Content-Length=[88]
Date=[Sat, 05 Sep 2015 15:35:43 GMT]
Content-Type=[text/plain]
X-Powered-By=[Undertow/1]
location = http://localhost:8080/vehicle/rs/vehicle/1
conflictInformation = conflict occured javax.persistence
```

- Jetzt funktionieren wieder alle Tests

Logger

Ziel

- Aufrufe alle Methoden in der Business-Schicht protokollieren. Die Business-Schicht wird von allen Technologien verwendet. Egal ob ein Zugriff per REST oder JSF oder … Es wird immer über die Facades auf die Daten zugegriffen

- Wir verwenden hierzu Interceptors, die bei jedem Methodenaufruf durchgeführt werden

https://www.voxxed.com/blog/2015/01/java-ee-interceptors/

```java
public class LogSinkProducer {

    @Produces
    public LogSink produce(InjectionPoint ip) {
        Class<?> injectionTarget = ip.getMember().getDeclaringClass();
        return Logger.getLogger(injectionTarget.getName())::info;
    }
}
```

Die aufrufende Klasse wird mitgegeben, wodurch es möglich ist, auf den aufrufenden Klassennamen zuzugreifen (vgl Goncalves, S.39)

```java
@FunctionalInterface
public interface LogSink {
    void log(String msg);
}
```

```java
public class BoundaryLogger {

    @Inject
    LogSink LOG;

    @AroundInvoke
    public Object logCall(InvocationContext ic) throws Exception {
        LOG.log("--" + ic.getMethod());
        return ic.proceed();
    }
}
```

Test-Client

Output des Loggers

Run at.htl.vehicle.rest in Vehicle-st

All 4

Test	Time
rest (at.htl.vehicle)	1s 479ms
VehicleResourceIT	1s 479ms
createVehicleWithInvalidTypeLength	972ms
crud	449ms
createValidVehicle	33ms
createVehicleWithoutBrand	25ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bir
Server=[WildFly/9]
validation-exception=[true]
Connection=[keep-alive]
Content-Length=[88]
Date=[Sat, 05 Sep 2015 15:55:49 GMT]
Content-Type=[text/plain]
X-Powered-By=[Undertow/1]
location = http://localhost:8080/vehicle/rs/vehicle/1
conflictInformation = conflict occured javax.persistence
```

Die ClientTests funktionieren alle, am Server sieht man nun die Logging Outputs

Output

```
    at org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:340)
    at org.jboss.as.ejb3.tx.CMTTxInterceptor.invokeInOurTx(CMTTxInterceptor.java:275)
    ... 86 more
Caused by: org.hibernate.StaleObjectStateException: Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect) : [at.htl.vehicle.entity.Vehicle#1] <7 internal calls>
    ... 126 more

17:55:50,439 INFO  [at.htl.vehicle.business.logging.BoundaryLogger] (default task-9) --public at.htl.vehicle.entity.Vehicle at.htl.vehicle.business.VehicleFacade.findById(long)
17:55:50,464 INFO  [at.htl.vehicle.business.logging.BoundaryLogger] (default task-10) --public at.htl.vehicle.entity.Vehicle at.htl.vehicle.business.VehicleFacade.updateVignetteValid(long,boolean)
17:55:50,478 INFO  [at.htl.vehicle.business.logging.BoundaryLogger] (default task-11) --public at.htl.vehicle.entity.Vehicle at.htl.vehicle.business.VehicleFacade.findById(long)
17:55:50,490 INFO  [at.htl.vehicle.business.logging.BoundaryLogger] (default task-12) --public at.htl.vehicle.entity.Vehicle at.htl.vehicle.business.VehicleFacade.updateVignetteValid(long,boolean)
17:55:50,506 INFO  [at.htl.vehicle.business.logging.BoundaryLogger] (default task-14) --public java.util.List at.htl.vehicle.business.VehicleFacade.findAll()
17:55:50,537 INFO  [at.htl.vehicle.business.logging.BoundaryLogger] (default task-15) --public void at.htl.vehicle.business.VehicleFacade.delete(long)
17:55:50,568 INFO  [at.htl.vehicle.business.logging.BoundaryLogger] (default task-16) --public at.htl.vehicle.entity.Vehicle at.htl.vehicle.business.VehicleFacade.save(at.htl.vehicle.entity.Vehicle)
```

Varianten für LogSink Producer

```java
public class LogSinkProducer {

    @Produces
    public LogSink produce(InjectionPoint ip) {
        Class<?> injectionTarget = ip.getMember().getDeclaringClass();

        Logger logger = Logger.getLogger(injectionTarget.getName());
        return new LogSink() {
            @Override
            public void log(String msg) {
                logger.info(msg);
            }
        };
    }
}
```

```java
public class LogSinkProducer {

    @Produces
    public LogSink produce(InjectionPoint ip) {
        Class<?> injectionTarget = ip.getMember().getDeclaringClass();
        Logger logger = Logger.getLogger(injectionTarget.getName());
        return (msg) -> logger.info(msg);
    }
}
```

```java
public class LogSinkProducer {

    @Produces
    public LogSink produce(InjectionPoint ip) {
        Class<?> injectionTarget = ip.getMember().getDeclaringClass();
        return Logger.getLogger(injectionTarget.getName())::info;
    }
}
```

Noch Fragen?