# Testdokumentation

(Anmerkung: Programm ist nicht lauffähig → No query defined for that name [team.findByName])

# Aufgabe 1: Import CSV

## readRacesFromFile() (InitBean.java):

- Einlesen der Datei "races.csv" und Speichern der Objekte in der Tabelle F1_RACE

## Falscher Code:

```
URL url = Thread.currentThread().getContextClassLoader() ①
                .getResource(RACES_FILE_NAME);
        try (Stream<String> stream = Files.lines(Paths.get(url.getPath()))) {
            DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.YYYY");
            LocalDate formatDate = LocalDate.parse(formatter);

            stream
                    .skip(1)
                    .map(s -> s.split(";"))
                    .map(a -> new Race(Long.getLong(a[1]), a[2], DateTimeFormatter
.formatDate(a[3]))
                    .forEach(em::merge);
                    //.forEach(System.out::println);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

- Ohne Lambda, DateTimeFormatter und LocalDate hat es funktioniert, also nur mit stream.forEach(System.out::println);. Eine Seite mit 'Forbidden' wurde ausgegeben.
- DateTimeFormatter gehört nicht in die try-catch.
- Das Format für das 3. Array (a[3]) ist falsch

## Richtiger Code:

```
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd.MM.yyyy");

        URL url = Thread.currentThread().getContextClassLoader()
                .getResource(racesFileName);
        try (Stream<String> stream = Files.lines(Paths.get(url.getPath().substring(3)
), StandardCharsets.UTF_8)) {
            stream
                    .skip(1) ①
                    .map((String s) -> s.split(";")) ②
                    .map(a -> new Race(Long.valueOf(a[0]), a[1], LocalDate.parse(a[2],
dtf))) ③
                    .forEach(em::persist); ④
                    //.forEach(System.out::println);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

- DateTimeFormatter gehört außerhalb der try-catch, weil es für das 3. Array (a[2]) benutzt wird.
- .subString(3) ...man liest die URL erst von der 3. Stelle (Windows hat Probleme mit dem Pfad C://, das Doppelpunkt war das Problem)
- StandardCharsets.UTF_8 ...für die Umlaute, falls es welche gibt

① Die erste Zeile von der CSV skippen

② Einzelne Daten trennen (mit dem ;)

③ :

- Das 1. Array (a[0]) hat einen Long Type (siehe Race Class), daher gehört davor ein Long.valueOf
- Das 2. Array (a[1]) hat einen String Type (siehe Race Class), daher kommt davor nichts
- Das 3. Array hat einen LocalDate Type (siehe Race Class), daher wird das LocalDate.parse verwendet. Innerhalb der Klammer wird das DateTimeFormatter dtf verwendet, weil die Dates in der CSV im Format "dd.MM.YYYY" eingetragen sind.

④ mit persist kann man die Daten in die DB einfügen

# readTeamsAndDriversFromFile() (InitBean.java):

- Einlesen der Datei "teams.csv".
- Das String-Array jeder einzelnen Zeile wird der Methode persistTeamAndDrivers(...) übergeben.

## Falscher Code:

```
URL url = Thread.currentThread().getContextClassLoader()
                .getResource(TEAM_FILE_NAME);
        try (Stream stream = Files.lines(Paths.get(url.getPath()))) {
            stream.forEach(System.out::println);
        } catch (IOException e) {
            e.printStackTrace();
        }

        new BufferedReader(new InputStreamReader(this.getClass()
                .getResourceAsStream(RACES_FILE_NAME), Charset.defaultCharset()))
                .lines()
                .skip(1)
                .map(s -> s.split(";"))
                .map(a -> new Race(a[1], a[2], a[3]))
                .forEach(em::merge);
```

## Richtiger Code:

```
URL url = Thread.currentThread().getContextClassLoader()
                .getResource(teamFileName);
        try (Stream<String> stream = Files.lines(Paths.get(url.getPath().subString(3)
), StandardCharsets.UTF_8)) {
            stream
                    .skip(1)
                    .map((String s) -> s.split(";"))
                    .forEach(this::persistTeamAndDrivers); ①
        } catch (IOException e) {
            e.printStackTrace();
        }
```

① Das String-Array jeder einzelnen Zeile wird der Methode persistTeamAndDrivers(...) übergeben. Damit die Daten in der Table angezeit werden, muss man die Methode persistTeamAndDrivers() erstellen, da man hier nur die CSV einliest, aber nicht abspeichert.

## persistTeamAndDrivers() (InitBean.java):

- Es wird überprüft ob es das übergebene Team schon in der Tabelle F1_TEAM gibt.
- Falls nicht, wird das Team in der Tabelle gespeichert.
- Wenn es das Team schon gibt, dann liest man das Team aus der Tabelle und erstellt ein Objekt (der Klasse Team).
- Dieses Objekt wird verwendet, um die Fahrer mit Ihrem jeweiligen Team in der Tabelle F1_DRIVER zu speichern.
- @param line String-Array mit den einzelnen Werten der csv-Datei

## Falscher Code:

(nicht gemacht)

## Richtiger Code:

```
Team team = null; ①

        try { ②
            team = em
                        .createNamedQuery("team.findByName", Team.class) ③
                        .setParameter("NAME", line[0]) ④
                        .getSingleResult(); ⑤
        } catch (NoResultException ex){ ⑥
            team = new Team(line[0]);
            em.persist(team);
        }

        Driver d1 = new Driver(line[1], team); ⑦
        Driver d2 = new Driver(line[2], team);
        em.persist(d1);
        em.persist(d2);
```

① team auf null setzen

② Es wird überprüft ob es das übergebene Team schon in der Tabelle F1_TEAM gibt. Falls nicht, wird das Team in der Tabelle gespeichert.

③ Dient zum Definieren von Abfragen mit Namen in Zuordnungsdateien oder Anmerkungen

④ Bevor die Query ausgeführt werden kann, muss ein Parameterwert mit der Methode setParameter festgelegt werden. Die Methode setParameter unterstützt die Verkettung von Methoden, sodass der Aufruf von getSingleResult mit demselben Ausdruck verkettet werden kann.

⑤ Queries ausführen, die ein einzelnes Ergebnisobjekt zurückgeben

⑥ Wenn es das Team schon gibt, dann liest man das Team aus der Tabelle und erstellt ein Objekt (der Klasse Team)

⑦ Dieses Objekt wird verwendet, um die Fahrer mit Ihrem jeweiligen Team in der Tabelle F1_DRIVER zu speichern

# ENTITIES:

## Driver.java

## Falscher Code:

```
@NamedQueries({
        //...
        @NamedQuery( ①
                name = "Race.findByNAme",
                query = "select d from Driver d where d.name = :NAME"
        )
})
```

① 2. NamedQuery ist falsch

## Richtiger Code:

```
@NamedQueries({
        @NamedQuery(
                name = "Driver.findAll",
                query = "select d from Driver d"
        ),
        @NamedQuery(
                name = "Driver.findByName", ①
                query = "select d from Driver d where d.name = :NAME" ②
        )
})

//...
@GeneratedValue(strategy = GenerationType.IDENTITY) ③
```

① Driver.

② :NAME wegen dem .setParameter("NAME", line[0]) bei der persistTeamAndDrivers() Methode

③ The id's are assigned by the database ...deshalb @GeneratedValue

# Race.java

## Falscher Code:

```
@NamedQueries({ ①
        @NamedQuery(
                name = "Race.findAll",
                query = "select r from Race r"
        )
        //...
})

//...
@GeneratedValue(strategy = GenerationType.IDENTITY) ②
```

① nicht nötig

② wird nicht benötigt, weil ...The id's are not assigned by the database. The id's are given.

## Richtiger Code

```
@NamedQuery(
        name = "Race.findByCountry",
        query = "select r from Race r where r.county = :COUNTRY"
)
```

# Team.java

## Falscher Code:

(nicht gemacht)

## Richtiger Code:

```
@Entity
@Table(name = "F1_TEAM")
@NamedQueries({
        @NamedQuery(
                name = "Team.findAll",
                query = "select t from Team t"
        ),
        @NamedQuery(
                name = "Team.findByName",
                query = "select t from Team t where t.name = :NAME"
        )
})
//...
@Id
```

# Result.java

(erst nach ResultEndpoint.java machen)

## Falscher Code:

(nicht gemacht)

## Richtiger Code:

```
@Entity
@Table(name = "F1_RESULT")
@NamedQueries({
        @NamedQuery(
                name = "Result.sumPointsForDriver", ①
                query = "select sum(r.points) from Result r where r.driver.name =
:NAME"
        ),
        @NamedQuery(
                name = "Result.findWinnerOfRace", ②
                query = "select r1.driver from Result r1 where r1.race = :RACE" +
                        "and r1.points >= all(select max(r2.points) from Result r2
where r2.race=r1.race)"
        ),
        @NamedQuery(
                name = "Result.findRacesWonByTeam", ③
                query = "select r.race from Result r where r.driver.team = :TEAM and
r.position=1"
        ),
        @NamedQuery(
                name = "Result.sumPointsForAllDrivers", ④
                query = "select r.driver.name, sum(r.points) from Result r group by
r.driver.name"
        )
})

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

① für getPointsSumOfDriver() im ResultsEndpoint.java

② für findWinnerOfRace() im ResultsEndpoint.java

③ für findRacesWonByTeam() im ResultsEndpoint.java

④ für getPointsSumOfAllDrivers() im ResultsEndpoint.java

# Aufgabe 2: Import REST

## readResultsFromEndpoint() (ResultsRestClient.java)

- Vom RestEndpoint werden alle Results abgeholt und in ein JsonArray gespeichert.
- Dieses JsonArray wird an die Methode persistResult(...) übergeben

## Falscher Code:

```
//...
List<JsonObject> jsonArray = JsonValue.getValueArray(JsonObject.class); ①
//...
```

① nicht nötig gewesen

## Richtiger Code:

```
@PersistenceContext ①
EntityManager em;

//...
client = ClientBuilder.newClient();
        target = client.target(RESULTS_ENDPOINT);

        Response response = this.target
                .request(MediaType.APPLICATION_JSON)
                .get();
        JsonArray payload = response.readEntity(JsonArray.class);
        persistResult(payload);
```

① hat außerhalb der Methode gefehlt

## persistResult() (ResultsRestClient.java)

- Das JsonArray wird durchlaufen (iteriert). Man erhält dabei Objekte vom Typ JsonValue. diese werden mit der Methode .asJsonObject() in ein JsonObject umgewandelt.
- Mit den entsprechenden get-Methoden können nun die einzelnen Werte (raceNo, position und driverFullName) ausgelesen werden.
- Mit dem driverFullName wird der entsprechende Driver aus der Datenbank ausgelesen.
- Dieser Driver wird dann dem neu erstellten Result-Objekt übergeben

## Falscher Code:

```
JsonObject jsonObj = new JsonObject(json.get("msg").toString());
for (int i = 0; i < resultsJson.length(); i++){
    JsonArray object = (JsonArray) resultsJson.getJsonObject(i);
    System.out.println(jsonObj.getString("body"));
}
```

## Richtiger Code:

```
for (JsonValue jsonValue : resultsJson) { ①
    JsonObject resultJson = jsonValue.asJsonObject();
    Driver driver = em ②
                    .createNamedQuery("Driver.findByName", Driver.class)
                    .setParameter("NAME", resultJson.getString("driverFullName")) ③
                    .getSingleResult();

    Race race = em ②
                    .createQuery("select r from Race r where r.id = :ID", Race.class)
                    .setParameter("ID", Long.valueOf(resultJson.getInt("raceNo")))
                    .getSingleResult();

    Result result = new Result(
                    race,
                    resultJson.getInt("position"),
                    driver);

    //Punkte vom Driver an einer gewissen Position kriegen
    result.setPoints(result.pointsPerPosition[result.getPosition()]); ⑤

    em.persist(result);
}
```

① Das JsonArray wird durchlaufen (iteriert). Man erhält dabei Objekte vom Typ JsonValue. diese werden mit der Methode .asJsonObject() in ein JsonObject umgewandelt.

② Die einzelnen Werte (raceNo, position und driverFullName) können ausgelesen werden.

③ Mit dem driverFullName wird der entsprechende Driver aus der Datenbank ausgelesen.

④ Dieser Driver wird dann dem neu erstellten Result-Objekt übergeben

⑤ Errechnen der Punkte

## ResultsEndpoint.java

(nicht gemacht)

## Errechnen der Punkte:

(nicht gemacht)

## persistResult() (ResultsRestClient.java)

- Punkte vom Driver an einer gewissen Position kriegen
- (siehe Result.java → public int[] pointsPerPosition)
- (siehe persistResult() von ResultsRestClient.java)

## Richtiger Code:

```
result.setPoints(result.pointsPerPosition[result.getPosition()]);
```

## Ermitteln der Id des Fahrers

(nicht gemacht)

- (siehe getPointsSumOfDriver in ResultsEndpoint.java)

# Aufgabe 3: Gesamtpunkte eines Fahrers

## getPointsSumOfDriver() (ResultsEndpoint):

## Falscher Code:

(nicht gemacht)

## Richtiger Code:

```
Driver driver = em ①
                .createNamedQuery("Driver.findByName", Driver.class)
                .setParameter("NAME", name)
                .getSingleResult();

Long points = em
                .createNamedQuery("Result.sumPointsForDriver", Long.class)
                .setParameter("NAME", name)
                .getSingleResult();

JsonObject jsonObj = Json.createObjectBuilder()
                .add("driver", driver.getName())
                .add("points", points)
                .build();

return jsonObj;
```

① Ermitteln der Id des Fahrers

# Aufgabe 4: Sieger eines bestimmten Rennens

## findWinnerOfRace() (ResultsEndpoint):

### Falscher Code:

(nicht gemacht)

### Richtiger Code:

```
@GET
@Path("winner/{country}") ①
@Produces(MediaType.APPLICATION_JSON)
public Response findWinnerOfRace(@PathParam("country") String country) {
    Race race = em
                .createNamedQuery("Race.findByCountry", Race.class)
                .setParameter("COUNTRY" , country)
                .getSingleResult();

    Driver winnerDriver = em
                .createNamedQuery("Driver.findWinnerOfRace", Driver.class)
                .setParameter("RACE", race)
                .getSingleResult();

    return Response.ok(winnerDriver).build();
}
```

① In der Aufgabenstellung ist ein Request angegeben: GET http://localhost:8080/formula1/api/results/winner/Spain …im Pfad steht winner vor der Stadt, deshalb winner/{country}

# Aufgabe 5: Liste der Rennen, die ein Team gewonnen hat

## Falscher Code:

(nicht gemacht)

## Richtiger Code:

```
@GET
@Path("raceswon")
public Response findRacesWonByTeam(@QueryParam("team") String team){
        Team team1 = em
                .createNamedQuery("Team.findByName", Team.class)
                .setParameter("NAME", team)
                .getSingleResult();

        List<Race> racesWon = em
                .createNamedQuery("Race.findRacesWonByTeam", Race.class)
                .setParameter("TEAM", team)
                .getResultList();

        return Response.ok(racesWon).build();
}
```

# Aufgabe 6 (für Spezialisten): Liste aller Fahrer mit ihren Punkten

## Falscher Code:

(nicht gemacht)

## Richtiger Code:

```
 @GET
@Path("all")
@Produces(MediaType.APPLICATION_JSON)
public Response getPointsSumOfAllDrivers(){
        List<Object[]> points = em
                .createNamedQuery("Result.sumPointsForAllDrivers", Object[].class)
                .getResultList();

        return Response.ok(points).build();
}
```