Testdokumentation

Aufgabe 1: Import CSV

```
try {
            BufferedReader br = new BufferedReader(new InputStreamReader(getClass()
.getResourceAsStream("/teams.csv")));
            br.readLine();
            String line2;
            while ((line2 = br.readLine()) != null) {
                String[] rowCell = line2.split(";");
                Driver d = new Driver();
                Team t = new Team();
                d.setName(rowCell[1]);
                d.setName(rowCell[2]);
                em.persist(d);
                em.persist(new Team(rowCell[0]));
                /*List<Driver> driver =
this.em.createNamedQuery("Driver.getDriverByName", Driver.class) ①
                        .setParameter("NAME", rowCell[1])
                        .setParameter("NAME", rowCell[2])
                        .getResultList();
                Driver currentDriver;
                /*if (driver.size() != 1) {
                    currentDriver = new Driver(currentDriver.setName(rowCell[1]));
                    em.persist(currentDriver);
                } else {
                    currentDriver = driver.get(0);
                }*/
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Ich habe eine andere Variante versucht, weil ich mit der schon geübt habe, bevor die Empfehlung von Ihnen kam Variante 4 zu verwenden. Ich habe die Daten auch nicht an die Methode "persistTeamAndDrivers" weitergegeben sondern versucht die Daten sofort zu speichern. Auch das richtige Splitten auf die zwei verschiedenen Tabellen in der Datenbank ist mir nicht richtig gelungen, auch wenn ich einen richtigen Ansatz (named query) <1> hatte auf den ich leider nicht genug vertraut hatte.

Hier der verbesserte Code:

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

Die Methode persistTeamAndDrivers sollte wie folgt aussehen:

```
    private void persistTeamAndDrivers(String[] line) {
        Team team = null;
        try {
            team = em.createNamedQuery("Team.findByName", Team.class)
                    .setParameter("NAME", line[0])
                    .getSingleResult();
        } catch (NoResultException ex) {
            team = new Team(line[0]);
            em.persist(team);
        }
        em.persist(new Driver(line[1], team));
        em.persist(new Driver(line[2], team));
    }
```

Bei den Entity-Klassen habe ich auch die Table-Annotation vergessen. Diese sieht beispielsweise für die Entität Driver so aus:

```
@Table(name = "F1_DRIVER")
```

Aufgabe 2: Import REST

```
public void readResultsFromEndpoint() {
        Response response = this.target.request(MediaType.APPLICATION_JSON).get();
        JsonArray payload = response.readEntity(JsonArray.class);
        /*for (JsonValue item : payload) {
            JsonObject resultJson = Json.createObjectBuilder().build();
            resultJson.
        }*/
        JsonObject object = payload.getJsonObject(0);
        persistResult(payload);
    }

@Transactional
void persistResult(JsonArray resultsJson) {
        JsonObject object = resultsJson.getJsonObject(0);
        List<JsonObject> values = resultsJson.getValuesAs(JsonObject.class);
        for (JsonObject value : values) {
            System.out.println(value);
            //JsonObject resultJson = Json.createObjectBuilder().build();
            //Race race = new Race(race.getId(), race.getCountry(),
value.getString("raceNo"));
            Team team = new Team();
            Driver driver = new Driver(value.getString("driverFullName"), team);
            //em.persist(new Result(race, value.getInt("position"), driver));
        }
    }
```

Hier war ich so weit, dass ich mir die Daten vom REST-Service in der Konsole ausgeben lassen konnte, ich habe es aber nicht geschafft diese Daten richtig in die Datenbank zu speichern.

In der Methode readResultsFromEndpoint() habe ich eine unnötige Zeile Code geschrieben und zwar:

```
JsonObject object = payload.getJsonObject(0);
```

Der verbesserte Code der Methode persistResult(JsonArray resultsJson) sieht wie folgt aus:

```
    @Transactional
    void persistResult(JsonArray resultsJson) {
        for (JsonValue value : resultsJson) {
            JsonObject resultJson = value.asJsonObject();
            Driver driver = em
                    .createNamedQuery("Driver.getDriverByName", Driver.class)
                    .setParameter("NAME", resultJson.getString("driverFullName"))
                    .getSingleResult();
            Race race = em
                    .createQuery("select r from Race r where r.id = :ID", Race.class)
                    .setParameter("ID", Long.valueOf(resultJson.getInt("raceNo")))
                    .getSingleResult();
            em.persist(new Result(race, resultJson.getInt("position"), driver));
        }
    }
```

Die folgenden Aufgaben konnte ich leider nicht lösen

Aufgabe 3: Gesamtpunkte eines Fahrers

Hierzu bin ich gar nicht mehr gekommen, weil ich keine Daten in der Tabelle Result hatte. Jetzt habe ich eine NamedQuery hinzugefügt, welche die Gesamtpunktezahl für einen bestimmten Fahrer zurückgibt.

```
        @NamedQuery(
                name = "Result.sumPointsForDriver",
                query = "select sum(r.points) from Result r where r.driver.name =
:NAME"
        ),
```

Zuerst muss ich in der Endpoint-Klasse einen EntityManager erstellen.

```
@PersistenceContext
EntityManager em;
```

Zusätzlich muss noch die Methode getSumPointsOfDriver im ResultsEndpoint implementiert werden.

Das sieht dann wie folgt aus:

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public JsonObject getPointsSumOfDriver(
            @QueryParam("name") String name
    ) {
        Long points = em
                .createNamedQuery("Result.sumPointsForDriver", Long.class)
                .setParameter("NAME", name)
                .getSingleResult();
        Driver driver = em
                .createNamedQuery("Driver.getDriverByName", Driver.class)
                .setParameter("NAME", name)
                .getSingleResult();
        return Json.createObjectBuilder()
                .add("driver", driver.getName())
                .add("points", points)
                .build();
    }
```

Aufgabe 4: : Sieger eines bestimmten Rennens

Um diese Aufgabe zu lösen muss zuerst die Methode erstellt werden.

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("winner/{country}")
    public Response findWinnerOfRace(@PathParam("country") String country) {

    }
```

Dann muss man eine Query in der Klasse Result hinzufügen, die den Driver zurückgibt, der auf Position 1 gelandet ist und überprüft ob das mitgegebene Land übereinstimmt.

```
@NamedQuery(
name = "Result.getWinnerOfRace",
query = "select re.driver from Result re where re.position = 1 and " +
                        "re.race = (select ra.id from Race ra where ra.country like
:COUNTRY)"
)
```

In der Klasse im Endpoint wird nun nach der ID des Siegers eines bestimmten Rennens gesucht und mit dieser ID, dann einer Objekt der Klasse Driver erstellt, welches zurückgegeben wird.

So sieht die fertige Methode im Endpoint aus:

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("winner/{country}")
    public Response findWinnerOfRace(@PathParam("country") String country) {
        Long driverId = em.createNamedQuery("Result.getWinnerOfRace", Driver.class)
                .setParameter("COUNTRY", country)
                .getSingleResult()
                .getId();
        Driver winner = em.find(Driver.class, driverId);
        return Response.ok(winner).build();
    }
```

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

Auch hier muss man zuerst die Methode erstellen.

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("raceswon")
    public List<Race> racesWonByTeam(@QueryParam("team") String team) {

    }
```

Der nächste Schritt erfolgt wieder mit dem erstellen der NamedQuery. Hier werden alle gewonnenen Rennen, des mitgegebenen Teams zurückgegeben.

```
@NamedQuery(
                name = "Result.racesWonByTeam",
                query = "select r.race from Result r where r.position = 1 " +
                        "and r.driver in (select distinct d.id from Driver d " +
                        "where d.team = (select distinct t.id from Team t where t.name
= :TEAM))"
        )
```

Zum Schluss wird die zuvor erstellte Methode racesWonByTeam(@QueryParam("team") String team) implementiert. Hier werden die Ergebnisse der Query in eine Liste von Races geschrieben und anschließend returned.

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("raceswon")
    public List<Race> racesWonByTeam(@QueryParam("team") String team) {
        List<Race> wonRaces = em.createNamedQuery("Result.racesWonByTeam", Race.class)
                .setParameter("TEAM", team)
                .getResultList();
        return wonRaces;
    }
```

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

Wieder beginnt man mit dem Erstellen der Methode.

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("all")
    public List<String[]> allDriversWithPoints() {

    }
```

Um diese Aufgabe zu lösen müssen diesmal zwei Querys implementiert werden.

```
@NamedQuery(
                name = "Driver.getDriver",
                query = "select d from Driver d"
        )
```

```
@NamedQuery(
                name = "Result.getPoints",
                query = "select sum(r.points) from Result r where r.driver = :ID"
        )
```

Um die Aufgabe nun zu lösen muss man die Methode richtig implementieren. Zuerst erstellt man eine List vom Typ Driver und speichert alle Driver hinein, diese werden von der Query zurückgegeben. Als nächstes erstellt man eine List vom Typ String-Array. Jetzt wird mit Hilfe einer For-Schleife durch die Liste drivers durchiteriert. Innerhalb der For-Schleife erstellt man eine Variable points vom Typ Long, in die das Ergebnis der zweiten Query gespeichert wird. Am Ende fügt man ein neues Objekt zur Liste hinzu. Außerhalb der Schleife wird diese Liste mit den String-Arrays returned.

```
List<Driver> drivers = em
                .createNamedQuery("Driver.getDriver", Driver.class)
                .getResultList();
        List<String[]> driversWithPoints = new LinkedList<>();
        for (Driver driver : drivers) {
            Long points = em.createNamedQuery("Result.getPoints", Long.class)
                    .setParameter("ID", driver)
                    .getSingleResult();
            driversWithPoints.add(new String[]{driver.toString(), "" + points});
        }
        return driversWithPoints;
```