

## UE4 –Web Services, Social Media, Advanced JPA (25 Punkte)

In der dritten Übung haben Sie eine serverseitige Lösung auf Basis des Play Frameworks implementiert. Ziel dieses Übungsbeispiels ist es nun, diese Implementierung zu erweitern und um eine Anbindung an das BIG Highscore Web Service bzw. an Twitter zu erweitern. Des Weiteren soll die Verwaltung von Quizfragen und –kategorien innerhalb einer Datenbank ermöglicht werden.

Deadline der Abgabe via TUWEL<sup>1</sup>: **Sonntag, 31. Mai 2015 23:55 Uhr**

### Jeopardy-Spiel

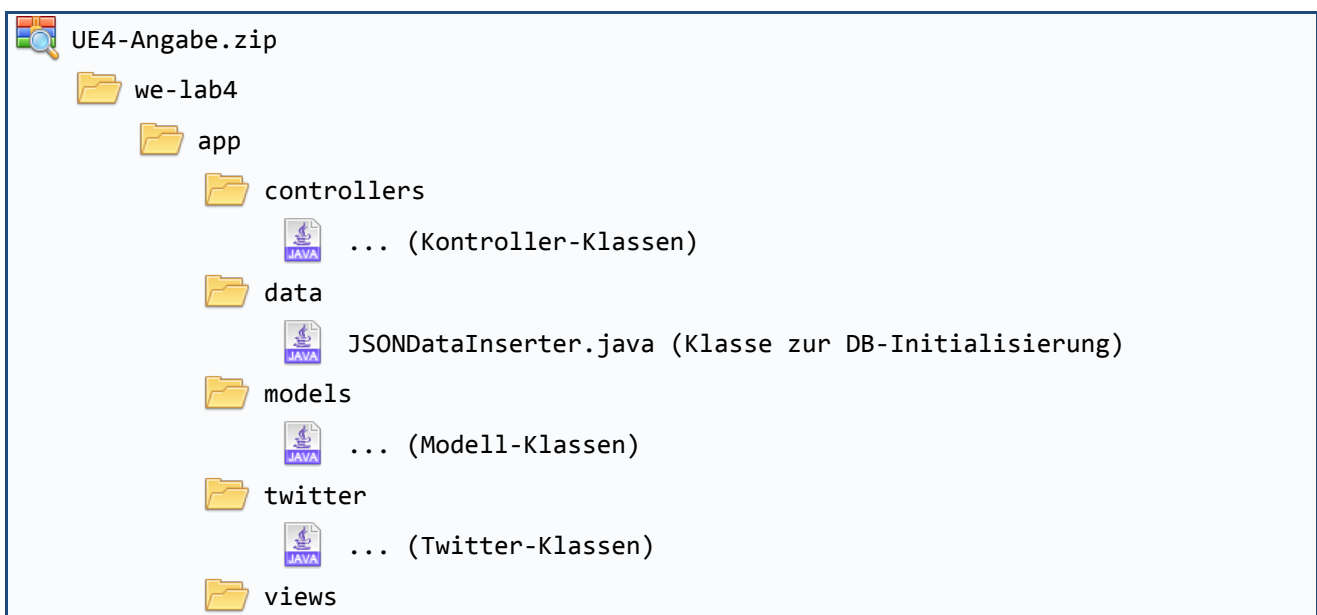
Bei „Jeopardy“ handelt es sich um ein Fragespiel, bei dem die KandidatInnen Fragen beantworten müssen, um Punkte zu sammeln. Jede Frage ist dabei einer Themenkategorie zugeordnet und hat einen bestimmten Wert, der den Schwierigkeitsgrad der Frage widerspiegelt. Eine Frage kann beliebig viele, aber mindestens zwei, Antwortmöglichkeiten haben, wobei mindestens eine dieser Antwortmöglichkeiten die korrekte Lösung auf die gestellte Frage ist („Multiple Choice“).

Ein Spiel besteht aus 10 Runden, wobei in jeder Runde eine Frage pro SpielerIn beantwortet wird. Der/Die SpielerIn (Mensch oder Computer) mit weniger Punkten darf zuerst eine Frage auswählen, bei Gleichstand wird der Mensch bevorzugt. Eine Frage gilt nur dann als richtig beantwortet, wenn alle korrekten Antwortmöglichkeiten ausgewählt und als Antwort an den Server übermittelt wurden. Zusätzlich gibt es bei jeder Frage ein Zeitlimit, bis zu dem diese Frage beantwortet werden muss. Ist dieses Zeitlimit abgelaufen, wird die aktuell ausgewählte Antwort gewertet.

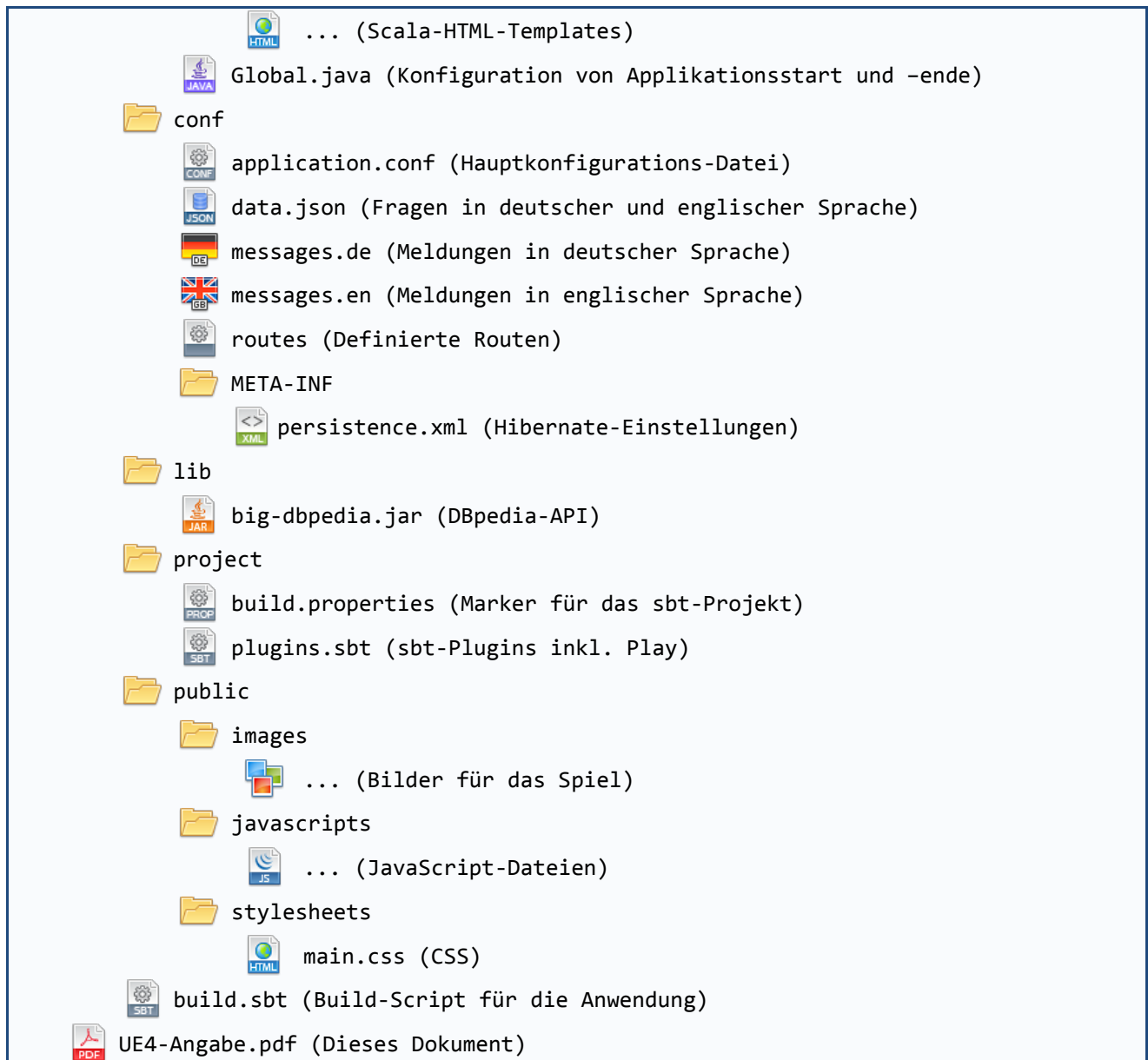
Wurden von beiden SpielerInnen alle Fragen beantwortet, wird der Punktestand beider SpielerInnen ausgewertet und der/die GewinnerIn angezeigt. Herrscht Gleichstand, gewinnt der Computer.

### Angabe

Diese Angabe umfasst folgende Dateien:



<sup>1</sup> <https://tuwel.tuwien.ac.at/course/view.php?id=6324>



## Teil 1 – Highscore-Service

Nach erfolgreichem Abschluss eines Spiels sollen die Spielergebnisse automatisch auf einem Highscoreboard gepostet werden. Für jedes Spiel soll dabei jeweils genau einmal das Highscoreergebnis gespeichert werden. Highscoreergebnisse können mittels eines SOAP/WSDL Web Services gepostet werden.

### URL des Highscoreboards:

<http://playground.big.tuwien.ac.at:8080/highscoreservice/>

### URL des Highscore Web Services:

<http://playground.big.tuwien.ac.at:8080/highscoreservice/PublishHighScoreService?wsdl>

### User-Key:

3ke93-gue34-dkeu9

Beachten Sie die folgenden Hinweise für die erfolgreiche Übermittlung eines Requests.

- Ein Servicerequest wird nur dann verarbeitet, wenn Sie im `<UserKey>` Element den korrekten UserKey angeben.
- Das `<Password>`-Element muss leer bleiben.
- Im `<FirstName>`-Element wird der Vorname des Spielers übermittelt.
- Im `<LastName>`-Element wird der Nachname des Spielers übermittelt.
- Im `<Points>`-Element werden die erreichten Punkte übertragen.
- Im `<BirthDate>`-Attribut wird das Geburtsdatum des Spielers übermittelt.
- Im `<Gender>`-Attribut wird das Geburtsdatum des Spielers übermittelt.
- Für die einzelnen Werte eines Benutzers übernehmen Sie die Daten, welche bei der Anmeldung des Benutzers erfasst wurden.
- Für den zweiten Benutzer (= Ihr PC) machen Sie sinnvolle Annahmen.

Wenn Sie alle Elemente und Attribute korrekt übergeben haben, bekommen Sie im Response eine UUID retourniert, welche Sie dann für die nächste Teilaufgabe benötigen. Wenn Sie fehlerhafte Daten übergeben, werden entsprechende Fehlermeldungen vom Service retourniert.

#### Beispielrequest:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:data="http://big.tuwien.ac.at/we/highscore/data">
  <soapenv:Header/>
  <soapenv:Body>
    <data:HighScoreRequest>
      <data:UserKey>3ke93-gue34-dkeu9</data:UserKey>
      <data:UserData>
        <Loser Gender="male" BirthDate="1990-01-12">
          <FirstName>Hans</FirstName>
          <LastName>Mustermann</LastName>
          <Password></Password>
          <Points>12</Points>
        </Loser>
        <Winner Gender="female" BirthDate="1981-01-12">
          <FirstName>Gerda</FirstName>
          <LastName>Haydn</LastName>
          <Password></Password>
          <Points>12</Points>
        </Winner>
      </data:UserData>
    </data:HighScoreRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

#### Zusammenfassung der geforderten Funktionalität von Teil 1:

- Highscoreergebnisse müssen auf dem Highscoreboard veröffentlicht werden und die UUID muss korrekt retourniert werden.
- Damit man das korrekte Ansprechen des Highscore-Services nachvollziehen kann, muss die retournierte UUID als Logging-Ausgabe ausgegeben werden. Benutzen Sie für das Logging die Klasse `play.Logger`. Loggen Sie sowohl im Erfolgs-, als auch im Fehlerfall.
- Für den Fall, dass das Highscore-Service nicht erreichbar ist, muss ein entsprechendes Exception Handling vorhanden sein (der User darf zB keinen HTTP 500 Fehler sehen).

## Teil 2 – Twitter Integration

Die in Aufgabe 1 gewonnene UUID soll in diesem Schritt per Aufruf der Twitter API als Tweet zu Twitter geschickt werden. Dabei sind folgende zwei Java Klassen verpflichtend zu verwenden:

- `TwitterStatusMessage.java`
- `ITwitterClient.java`

Der Text, welcher als Tweet auf Twitter gepostet werden soll, wird durch Aufruf der Methode

```
public String getTwitterPublicationString()
```

in der Klasse `TwitterStatusMessage` retourniert. Ihr `TwitterClient` muss das Interface `ITwitterClient` implementieren.

**Twitter URL:** <https://twitter.com/BIGewa2013> (kein Rechtschreibfehler - es wird der Twitter-Feed von 2013 verwendet)

**Consumer Key:**

GZ6tiy1XyB9W0P4xEJudQ

**Consumer Secret:**

gaJdlW0vf7en46JwHAOkZsTHvtAiZ3QUd2mD1x26J9w

**Access Token:**

1366513208-MutXEbBMAVOwrbFmZtj1r4lh2vcoHGHE2207002

**Access Token Secret:**

RMPWoePlus3xtURWRVnv1TgrjTyK7Zk33evp4KKyA

### Zusammenfassung der geforderten Funktionalität von Teil 2:

- Die UUID, welche durch den Aufruf des Highscore-Service gewonnen wurde, soll als Tweet auf Twitter veröffentlicht werden
- Nach erfolgreichem Aufruf des Highscore-Services und dem erfolgreichem Posten der UUID auf Twitter, soll dem Benutzer eine Statusmeldung „UUID xyz wurde auf Twitter veröffentlicht“ angezeigt werden.
- Damit man das korrekte Ansprechen der Twitter API nachvollziehen kann, muss eine Logging-Ausgabe ausgegeben werden. Benutzen Sie für das Logging die Klasse `play.Logger`. Loggen Sie sowohl im Erfolgs-, als auch im Fehlerfall.
- Für den Fall, dass das Publizieren auf Twitter fehlgeschlagen ist, muss ein entsprechendes Exception-Handling vorhanden sein (der User darf zB keinen HTTP 500 Fehler sehen)

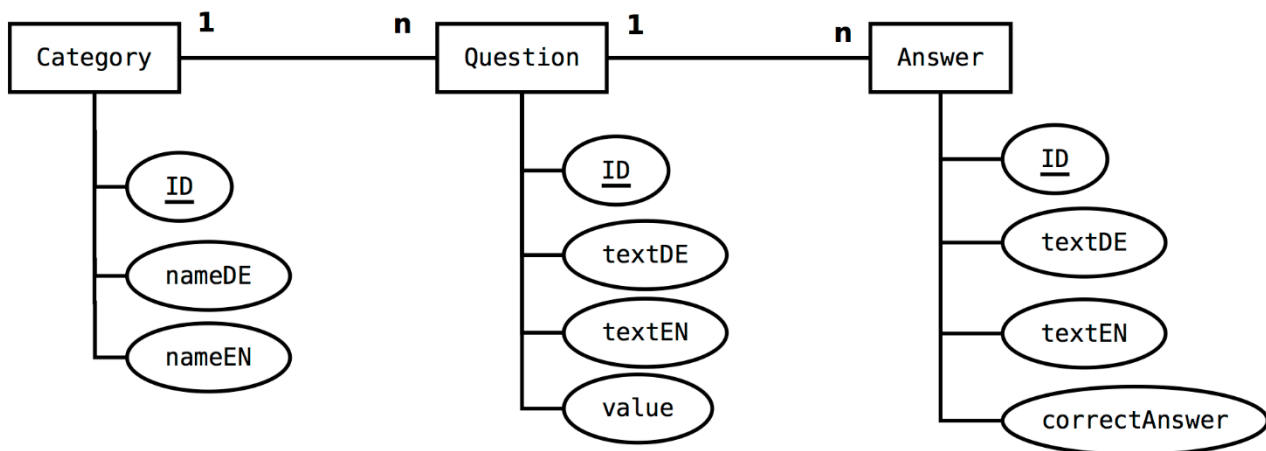
## Teil 3 – Advanced JPA

Bisher haben Sie die einzelnen Kategorien, Fragen und Antworten aus JSON Dateien bezogen. In dieser Teilaufgabe sollen Sie die bestehende Datenbank erweitern und die Möglichkeit schaffen, dass auch Kategorien, Fragen und Antworten gespeichert werden können. Im Unterordner `/app/models` finden Sie die folgenden vier Entityklassen:

- `BaseEntity`
- `Category`
- `Question`
- `Answer`

### Annotation der Entityklassen und Realisierung des DAO Interfaces

Annotieren Sie die oben genannten vier Entityklassen mit den korrekten JPA Annotationen, sodass die folgende DB-Struktur erfüllt ist.



Im Anschluss implementieren Sie das Interface IGameDAO in der Klasse JeopardyDAO und realisieren die konkreten Implementierungen der Methoden. Es wird empfohlen, dass Sie die erzeugten Entityklassen und die implementierten Methoden zuerst mit Hilfe von Unittests testen, bevor Sie diese in Ihre Anwendung einbauen.

### Teil 4 – Linked Open Data

In der Ihnen zur Verfügung gestellten Applikation werden bei Applikationsstart die Fragen aus der JSON-Datei (data.json) gelesen und in die konfigurierte Datenbank gespeichert (siehe Global.java und JSONDataInserter.java). Hierfür wird die von Ihnen zu implementierende persist-Methode des DAO-Objekts (JeopardyDAO) verwendet.

Vor Beginn jedes Spiels werden alle Kategorien samt Fragen und Antwortmöglichkeiten aus der Datenbank gelesen und in einem Spiel-Objekt (JeopardyGame) gespeichert, welches den Status des aktuellen Spiels beinhaltet.

Erweitern Sie nun den Datenbestand in der Datenbank, in dem Sie nachdem die JSON-Datei eingelesen wurde, selbständig Fragen auf Basis von DBpedia-Daten erzeugen und in die Datenbank speichern. Ziel ist es, dass Sie 5 unterschiedliche Fragen samt falscher und richtiger Antwortmöglichkeiten zu einer bestimmten Kategorie erzeugen. Die Auswahl der Kategorie ist Ihnen dabei frei gestellt und das Thema muss auch nichts mit Informatik zu tun haben. Achten Sie jedoch darauf, dieselben Fragen sowohl auf Deutsch als auch auf Englisch zu erstellen.

### Hinweise

#### Einbinden von DBpedia

Das DBpedia Projekt<sup>2</sup> stellt Daten aus Wikipedia als Linked Open Data (LOD) zur Verfügung. Die Daten werden dabei aus Wikipedia extrahiert, klassifiziert und entsprechend den LOD-Prinzipien aufbereitet und miteinander verlinkt. Über einen SPARQL Endpoint kann man auf diese Daten zugreifen<sup>3</sup>.

<sup>2</sup> <http://dbpedia.org>

<sup>3</sup> <http://wiki.dbpedia.org/OnlineAccess>

Grundsätzlich werden diese Daten als RDF-Graphen retourniert. Ein RDF-Graph besteht aus einer Menge von Statements, die aus den Tripeln Subject-Prädikat-Object bestehen.

Beispielsweise würde die folgende SPARQL-Abfrage<sup>4</sup> alle Subjekte retournieren, für welches ein Statement mit dem Prädikat `subject` und dem den Wert `Category:French_films` für dieses Prädikat existiert. Das Prädikat `subject` ist in dem Vokabular von PURL definiert, die Kategorie französischer Filme ist durch ein Vokabular in DBPedia definiert.

```
SELECT ?film
WHERE { ?film
  <http://purl.org/dc/terms/subject>
  <http://dbpedia.org/resource/Category:French\_films> }
```

Sie können Ihre Abfragen unter <http://dbpedia.org/sparql> testen.

Um Ihnen den Zugriff auf die DBpedia-Daten zu erleichtern, stellen wir Ihnen eine API zur Verfügung, die im Wesentlichen aus zwei Klassen (`DBpediaService`, `SelectQueryBuilder`) sowie verschiedenen Vokabulardefinitionen (zB Skos oder `DBpediaOWL`) besteht. Diese API basiert auf dem Apache Jena Framework, welches einerseits noch weitere Vokabulardefinitionen (zB FOAF oder RDFS) aber auch das Datenmodell für RDF-Graphen zur Verfügung stellt. RDF-Graphen sind in Jena als Model definiert, Statements als Statement, Subjekte als Resource, Prädikate als Property und Objekte als `RDFNode` (Super-Klasse von Resource, aber auch Literal-Werten, zB Zahlen oder Strings).

Die `DBpediaService`-Klasse bietet unter anderem Möglichkeiten um die Verfügbarkeit von DBPedia zu überprüfen (`isAvailable`), SPARQL-Abfragen auszuführen (`loadStatements`), den Namen von Ressourcen zu ermitteln (`getResourceName`) und Modelle auszugeben (`writeModel`).

Die `SelectQueryBuilder`-Klasse kann verwendet werden um SPARQL-Abfragen aufzubauen. Dabei kann man folgende Kriterien definieren:

WHERE: Statements, die für die gesuchten Subjekte/Ressourcen vorhanden sein müssen.

MINUS: Statements, die für die gesuchten Subjekte/Ressourcen nicht vorhanden sein dürfen.

FILTER: Prädikate/Property, die bestimmte Werte (Objekt/`RDFNode`) aufweisen müssen.

`PredicateExists`: Ein Prädikat muss vorhanden sein, egal welchen Wert es hat.

`PredicateNotExists`: Ein Prädikat darf nicht vorhanden sein.

Der `SelectQueryBuilder` stellt jedoch nicht die gesamte Funktionalität von SPARQL zur Verfügung (zB fehlende Aggregationsfunktionen). Sie können daher die `DBpediaService`-Klasse auch direkt mit SPARQL-Abfrage-Strings verwenden.

Für die genauere Verwendung berücksichtigen Sie bitte die JavaDoc-Kommentare in den entsprechenden Klassen und Methoden.

Bitte beachten Sie, dass die Daten von DBPedia automatisch aus Wikipedia verarbeitet werden. Dadurch kann es vorkommen, dass einige Datensätze unvollständig sind, insbesondere was den Ressourcenamen in verschiedenen Sprachen betrifft (abfragbar durch `getResourceName` bzw. `getResourceNames`). Die zur Verfügung gestellte API prüft die Prädikate `foaf:name` und `rdfs:label`. Es liegt daher an Ihnen, darauf zu achten, dass auf jeden Fall korrekte englische und deutsche Fragen und Antwortmöglichkeiten generiert werden.

<sup>4</sup> <http://www.w3.org/TR/sparql11-query/>

*Beispiel: Man möchte alle Filme, in denen Johnny Depp gespielt und Tim Burton Regie geführt hat, finden. Danach alle Filme, in denen Johnny Depp gespielt hat, Tim Burton aber nicht Regie geführt hat. Die Namen aller Filme werden sowohl in Deutsch als auch in Englisch benötigt (Überprüfung der Namen nicht enthalten).*

```
// Check if DBpedia is available
if(!DBpediaService.isAvailable())
    return;

// Resource Tim Burton is available at http://dbpedia.org/resource/Tim\_Burton
// Load all statements as we need to get the name later
Resource director = DBpediaService.loadStatements(DBpedia.createResource("Tim_Burton"));

// Resource Johnny Depp is available at http://dbpedia.org/resource/Johnny\_Depp
// Load all statements as we need to get the name later
Resource actor = DBpediaService.loadStatements(DBpedia.createResource("Johnny_Depp"));

// retrieve english and german names, might be used for question text
String englishDirectorName = DBpediaService.getResourceName(director, Locale.ENGLISH);
String germanDirectorName = DBpediaService.getResourceName(director, Locale.GERMAN);

String englishActorName = DBpediaService.getResourceName(actor, Locale.ENGLISH);
String germanActorName = DBpediaService.getResourceName(actor, Locale.GERMAN);

// build SPARQL-query
SelectQueryBuilder movieQuery = DBpediaService.createQueryBuilder()
    .setLimit(5) // at most five statements
    .addWhereClause(RDF.type, DBpediaOWL.Film)
    .addPredicateExistsClause(FOAF.name)
    .addWhereClause(DBpediaOWL.director, director)
    .addFilterClause(RDFS.label, Locale.GERMAN)
    .addFilterClause(RDFS.label, Locale.ENGLISH);

// retrieve data from dbpedia
Model timBurtonMovies = DBpediaService.loadStatements(movieQuery.toQueryString());

// get english and german movie names, e.g., for right choices
List<String> englishTimBurtonMovieNames =
    DBpediaService.getResourceNames(timBurtonMovies, Locale.ENGLISH);
List<String> germanTimBurtonMovieNames =
    DBpediaService.getResourceNames(timBurtonMovies, Locale.GERMAN);

// alter query to get movies without tim burton
movieQuery.removeWhereClause(DBpediaOWL.director, director);
movieQuery.addMinusClause(DBpediaOWL.director, director);

// retrieve data from dbpedia
Model noTimBurtonMovies = DBpediaService.loadStatements(movieQuery.toQueryString());

// get english and german movie names, e.g., for wrong choices
List<String> englishNoTimBurtonMovieNames =
    DBpediaService.getResourceNames(noTimBurtonMovies, Locale.ENGLISH);
List<String> germanNoTimBurtonMovieNames =
    DBpediaService.getResourceNames(noTimBurtonMovies, Locale.GERMAN);
```

## Abgabemodalität

Beachten Sie die allgemeinen Abgabemodalitäten des TUWEL-Kurses<sup>5</sup>. Zippen Sie Ihre Abgabe, sodass sie die folgende Struktur aufweist. **Alle Dateien müssen UTF-8 codiert sein!**

ACHTUNG: Wird das Abgabeschema nicht eingehalten, so kann es zu Punkteabzügen kommen!

<sup>5</sup> <https://tuwel.tuwien.ac.at/course/view.php?id=6324>

