

Information Retrieval Exercises

Assignment 1:

IMDB Spider and Queries

Samuele Garda (gardasam@informatik.hu-berlin.de)

IMDb: Internet Movie Database

THE 27:TH STOCKHOLM INTERNATIONAL FILM FESTIVAL NOVEMBER 9-20 2016 GUEST OF HONOR: FRANCIS FORD COPPOLA

IMDb Find Movies, TV shows, Celebrities and more... All

Movies, TV & Showtimes | Celebs, Events & Photos | News & Community | Watchlist

Unbegrenzter Film- und Seriengenuss mit Prime Instant Video Jetzt 30 Tage testen amazon

FULL CAST AND CREW | TRIVIA | USER REVIEWS | IMDbPro | MORE | SHARE

+ Avatar - Aufbruch nach Pandora (2009)

Avatar (original title)
12 | 2h 42min | Action, Adventure, Fantasy | 17 December 2009 (Germany)

7.9 / 10
898.519 Rate This

3:36 | Trailer 18 VIDEOS 242 IMAGES

Watch Now From EUR2.99 (SD) on Amazon Video ON DISC

A paraplegic marine dispatched to the moon Pandora on a unique mission becomes torn between following his orders and protecting the world he feels is his home.

Director: James Cameron
Writer: James Cameron
Stars: Sam Worthington, Zoe Saldana, Sigourney Weaver | See full cast & crew »

Metascore 83 From metacritic.com
Reviews 3.063 user | 720 critic
Popularity 332 (▲ 25)

Cast

Edit

Cast overview, first billed only:

	Sam Worthington	...	Jake Sully
	Zoe Saldana	...	Neytiri (as Zoë Saldana)
	Sigourney Weaver	...	Dr. Grace Augustine
	Stephen Lang	...	Colonel Miles Quaritch
	Michelle Rodriguez	...	Trudy Chacón
	Giovanni Ribisi	...	Parker Selfridge
	Joel David Moore	...	Norm Spellman
	CCH Pounder	...	Moat
	Wes Studi	...	Eytukan
	Laz Alonso	...	Tsu'tey
	Dileep Rao	...	Dr. Max Patel
	Matt Gerald	...	Corporal Lyle Wainfleet
	Sean Anthony Moran	...	Private Fike
	Jason Whyte	...	Cryo Vault Med Tech
	Scott Lawrence	...	Venture Star Crew Chief

See full cast »

Storyline

Edit

When his brother is killed in a robbery, paraplegic Marine Jake Sully decides to take his place in a mission on the distant world of Pandora. There he learns of greedy corporate figurehead Parker Selfridge's intentions of driving off the native humanoid "Na'vi" in order to mine for the

Assignment

- Task:
 - Given a list of 500 movie names, answer queries on such movies information
- Problem:
 - IMDB data is human-readable and semi-structured
- Pipeline:
 - “Scraping” data from IMDB, i.e.:
 - extracting data in structured format
 - Implementation of a Spider (a.k.a Web Crawler)
 - Perform queries on the scrapped data

Concrete tasks

1. Implement a JAVA program that reads a list of 500 movie titles from a JSON file
2. For each movie title, perform a web search on IMDB and retrieve movie's URL
3. For each movie, extract metadata (e.g. actors, plot, budget, description) from the movie's URL and store them in a JSON file
4. Implement specific queries on the data extracted

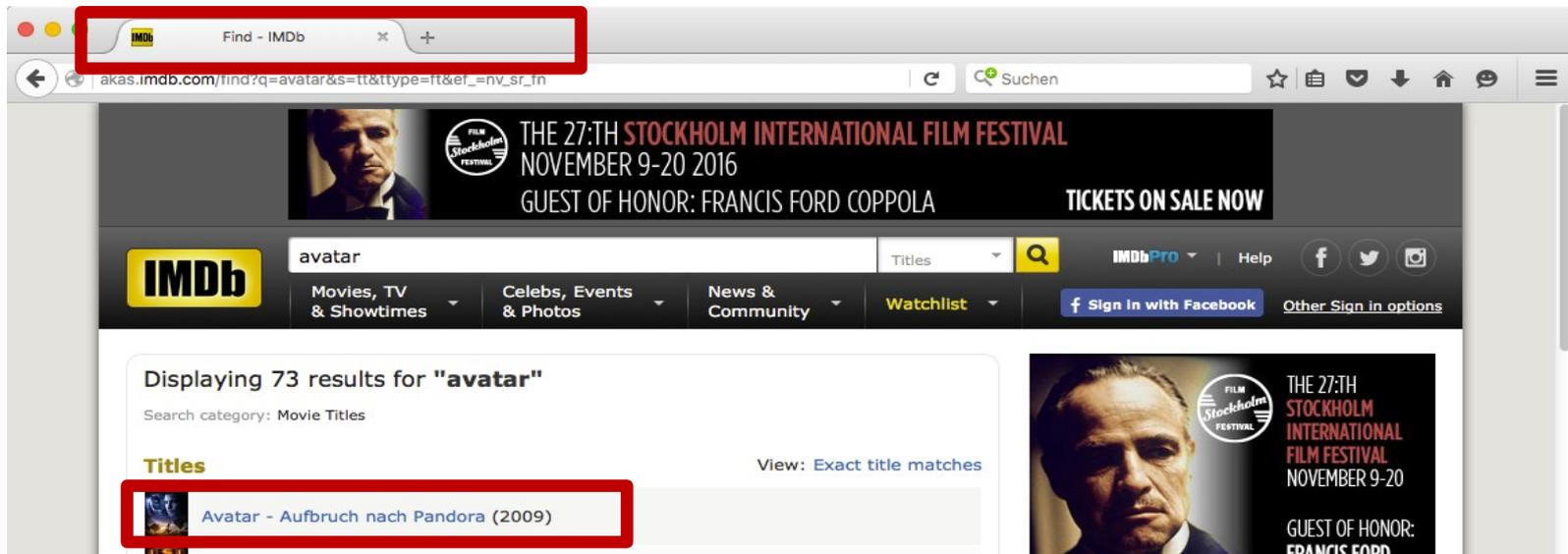
1. Read movie titles from JSON file

- Read movie titles from a JSON file “movies.json”:

```
[  
  {"movie_name": "Avatar"},  
  {"movie_name": "Star Wars VII: The Force Awakens"},  
  ...  
]
```
- You can use any Java library for parsing JSON files
 - Reference implementation: Oracle’s JSONP (<https://jsonp.java.net/>)
 - JSON.simple (<https://github.com/fangyidong/json-simple>)
 - GSON (<https://github.com/google/gson>)
 - Jackson Project (<https://github.com/FasterXML/jackson>)

2. IMDB Spider: getting the URL

- *The Spider IMDBSpider.java should open the URL:*
 - `https://www.imdb.com/find?q=<MOVIE>&s=tt&ttype=ft`
 - Where `<MOVIE>` is the movie name in the JSON file
- From the results, extract the first element and its URL
- Use URL encoding of movie titles



2. IMDB Spider: extracting movie url

- You have to parse the HTML file to extract the URL



The screenshot shows the IMDb website with the movie 'Avatar - Aufbruch nach Pandora' (2009) selected. The browser's developer tools are open, showing the HTML structure. The 'findList' table is highlighted, and the 'result_text' entry is selected. The URL for the movie page is visible in the href attribute of the selected entry.

The table is named „findList“

An entry is named „result_text“

3. Extract metadata from movie's URL

The image displays three overlapping browser windows showing the IMDb page for the movie 'Avatar - Aufbruch nach Pandora'. The top window shows the 'Cast' section, listing actors such as Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang, Michelle Rodriguez, Giovanni Ribisi, Joel David Moore, CCH Pounder, Wes Studi, Laz Alonso, Dileep Rao, Matt Gerald, Sean Anthony Moran, Jason Whyte, and Scott Lawrence. The middle window shows the 'Storyline' section, which includes a synopsis: 'When his brother is killed in a robbery, paraplegic Marine Jake Sully is sent on a mission on the distant world of Pandora. There he learns the intentions of driving off the native humans and recovering the precious material scattered throughout their rich woodland that will fix his legs, Jake gathers intel for the cooperating gung-ho Colonel Quaritch, while simultaneously attempting the use of an "avatar" identity. While Jake begins to bond with the beautiful alien Neytiri, the restless Colonel attempts extermination tactics, forcing the soldier to take a stand against the fate of Pandora. Written by The Massie Twins'. The bottom window shows the 'Technical Specs' section, listing details such as Runtime (162 min | 171 min (special edition) | 178 min (extended cut)), Sound Mix (Dolby Digital | DTS | SDDS | Sonics-DDP (IMAX version)), Color (Color), and Aspect Ratio (1.78 : 1). Other sections visible include 'Did You Know?', 'Quotes', 'Crazy Credits', 'Connections', 'Soundtracks', and 'Frequently Asked Questions'.

3. Extract metadata from Movie's URL

For each movie extract the following fields and store data into a JSON file:

- url
 - title
 - year
 - genreList
 - countryList
 - directorList
 - characterList
 - description
 - budget
 - gross
 - ratingValue
 - ratingCount
 - duration
 - castList
- Each attribute is:
 - A string, as url,title,year
 - A JSON list: *List
 - **Use exactly these names when storing into the JSON file!**
 - For an example output refer to *avatar.json*

3. Extract metadata from Movie's URL

- Sometimes one or more field can be missing:
 - Use empty strings or empty arrays!
- Special case "gross"
 - First search for "Cumulative Worldwide Gross"
 - If missing, search for "Gross USA"
 - Only use dollar values (no Yen, Euros, etc)!

Optional: extract further meta information:

- keywordList,
- aspectRatio
- contentRating
- reviews
- critics

3. Extract metadata from Movie's URL

- XPATH is a syntax for navigating parts of an XML document:
- Has a directory-path-like syntax
 - ```
<table class="list" >
 <tr>
 <td class="result">Avatar</td>
 </tr>
</table>
```
  - **`/table[@class='list']//td[@class='result']/text() → Avatar`**
- In Java:
  - `Htmlcleaner` (<http://htmlcleaner.sourceforge.net/>)
  - `Xsoup` (<https://github.com/code4craft/xsoup>)
  - `javax.xml.xpath.XPath`

# 4. Query the corpus

---

- Mandatory queries:
  - You have to correctly implement **at least**:
    - **three** basic queries
    - **two** hard queries
- Custom queries:
  - Come up with a fancy custom query
  - Give a text description of the query, the implementation and the result
  - Very creative queries can earn an extra point for the competition

# 4. Easy Queries I

---

## 1. All-rounder:

- all movies in which the director stars as an actor (cast)
- top ten matches sorted by decreasing IMDB rating.

## 2. Under the radar:

- the top ten US-American movies until (including) 2015 that have made the biggest **loss** despite an IMDB score above (excluding) 8.0, based on at least 1,000 votes.
- **loss** = gross - budget.

## 3. The pillars of storytelling:

- all movies that contain both (sub-)strings "kill" and "love" in their lowercase description (`String.toLowerCase()`).
- Sort the results by the number of appearances of these strings and return the top ten matches.

## 4. Easy Queries II

---

### 4. The red planet:

- Sci-Fi movies that mention "Mars" in their description (case sensitive!)
- List all found movies in ascending order of publication (year)

### 5. Colossal failure:

- all USA movies with:
  - a duration beyond 2 hours
  - a budget  $\geq 1$  million
  - IMDB rating below 5.0
- Sort results by ascending IMDB rating

# 4. Harder Queries (Aggregation & Join) I

---

## 1. Uncreative writers:

- ten most frequent character names of all times ordered by frequency of occurrence
- Filter any name containing "himself", "doctor", and "herself" from the result

## 2. Workhorse:

- top ten most active actors (cast), i.e., those actors which have starred in most movies (descending order)

## 3. Must see:

- best rated movie of each year starting from 1990 until (including) 2010 with more than 10,000 ratings.
- from oldest to most recent

## 4. Harder Queries (Aggregation & Join) II

---

### 4. Rotten Tomatoes:

- worst rated movie of each year starting from 1990 till (including) 2010 with an IMCB score larger than 0.
- from oldest to most recent

### 5. Magic Couples:

- Determine those couples that feature together in the most movies (e.g. Adam Sandler and Allen Covert feature together in multiple movies)
- top 10 couples and sort the result by the number of movies



# The code: IMDBSpider

```
public class IMDBSpider {
 public IMDBSpider() {
 //
 }
 public void fetchIMDBMovies(String movieListJSON, String outputDir) throws IOException {
 // TODO add code here
 }
 public static void main(String[] argv) throws IOException {
 String moviesPath = "./data/movies.json";
 String outputDir = "./data";

 if (argv.length == 2) {
 moviesPath = argv[0];
 outputDir = argv[1];
 } else if (argv.length != 0) {
 System.out.println("Call with: IMDBSpider.jar <moviesPath> <outputDir>");
 System.exit(0);
 }

 IMDBSpider sp = new IMDBSpider();
 sp.fetchIMDBMovies(movieListJSON, outputDir);
 }
}
```

DO NOT MODIFY!

EXACTLY 2 ARGUMENTS:  
- MOVIES NAMES  
- OUTPUT DIR

# The code: IMDBQueries

```
public class IMDBQueries {

 protected List<Tuple<Movie, String>> queryAllRounder(List<Movie> movies) {
 // TODO Basic Query: insert code here
 return new ArrayList<>();
 }
}
```

class Movie already provided (do not modify!)

```
public static void main(String[] argv) throws IOException {
 String moviesPath = "./data/movies/";

 if (argv.length == 1) {
 moviesPath = argv[0];
 } else if (argv.length != 0) {
 System.out.println("Call with: IMDBQueries.jar <moviesPath>");
 System.exit(0);
 }
}
```

DO NOT MODIFY THIS

```
MovieReader movieReader = new MovieReader();
List<Movie> movies = movieReader.readMoviesFrom(Paths.get(moviesPath));
```

EXACTLY ONE ARGUMENT: MOVIES DIR

```
System.out.println("All-rounder");
{
 IMDBQueries queries = new IMDBQueries();
 long time = System.currentTimeMillis();
 List<Tuple<Movie, String>> result = queries.queryAllRounder(movies);
 System.out.println("Time:" + (System.currentTimeMillis() - time));

 if (result != null && !result.isEmpty() && result.size() == 10) {
 for (Tuple<Movie, String> tuple : result) {
 System.out.println("\t" + tuple.first.getRatingValue() + "\t"
 + tuple.first.getTitle() + "\t" + tuple.second);
 }
 } else {
 System.out.println("Error? Or not implemented?");
 }
}
System.out.println("");
```

class MovieReader provided (do not modify!)

# Caveats

---

- Crawler:
  - *IMDBSpider.java*, which reads the movie titles from a JSON file and stores each movie in a single JSON file
- Queries:
  - You must implement five queries in *IMDBQueries.java*
  - Optional Custom Query: You can implement one fancy custom query. Give a description of the query, source code and the result.
  - **A query counts as implemented if it is correct.** So, implement more than five to be sure to complete the task!
  - No query result caching!

# Competition

---

- Queries should not only be correct but as fast as possible
- While you have 500 movies - I will execute your queries with 2500 movies
- Evaluation:
  - A correctly implemented query
  - Bonus for faster implementation

# Submission

---

- Java source codes and **two executable JARs**
  - `java -jar IMDBSpider.jar movies.json <moviesDir>`
  - `java -jar IMDBQueries.jar <moviesDir>`

# Submission

---

- Before submitting, make sure that you:
  - did not alter the functions' signatures (types of parameters, return values)
  - only use the default constructor and don't change its parameters
  - did not change the class or package name
  - named your jars "IMDBSpider.jar" and "IMDBQueries.jar"
  - tested your jar (on a gruenau)

# Submission

---

- Submission:
  - **Tuesday, 04.05., 23:59 (midnight)**
  - **Wednesday, 05.05., 23:59 (midnight)**
  
- Presentation:
  - Tuesday, 11.05.
  - Wednesday, 12.05

# Next week (attendance optional)

---

- Q/A session for assignment 1
- “Live” coding? Scraper example
- If you have questions about topics from the lecture, write me an email in advance!

---

# Questions?