Documentation

March 12, 2020

Contents

1 How-To use this program

Setup the database

Either run:

```
# Create the database tables
mysql <options> < src/main/resources/create_db.sql

# Enter testdata such that all use-cases can be tested
mysql <options> < src/main/resources/testdata.sql
```

1.1 How to run

The JAR file is built to be run using Java 11. Don't complain if you don't have Java 11.

```
java -jar IMDb2.jar <database_url> <database_username> <database_password>
```

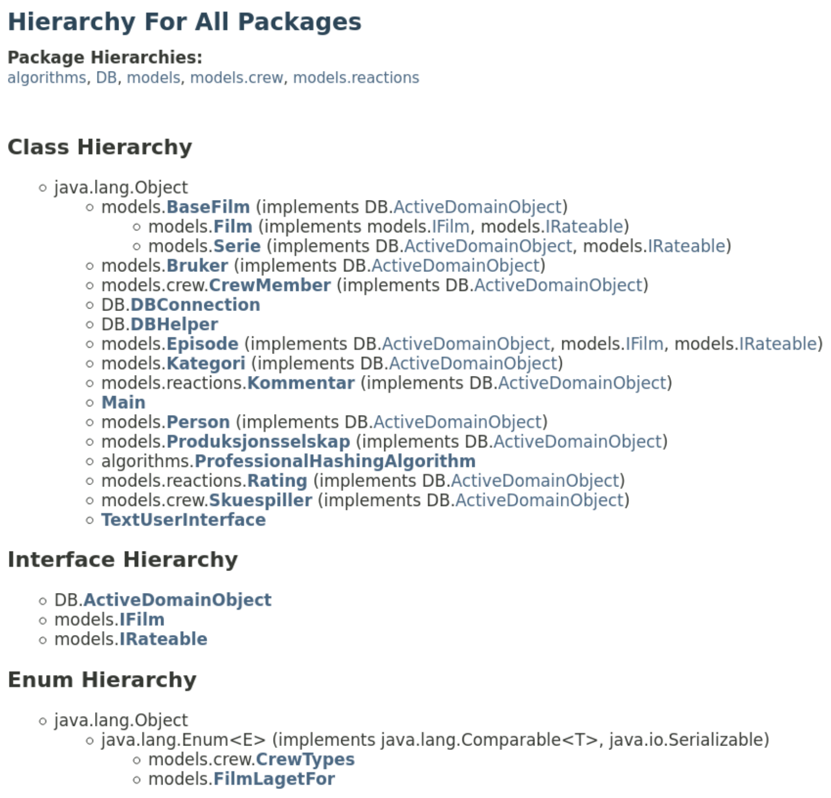
2 i) Overview and description of the classes

2.1 models.BaseFilm

BaseFilm contains the shared functionality between Film and Serie

2.2 models.BaseFilm

BaseFilm contains the shared functionality between Film and Serie



Hierarchy For All Packages

Package Hierarchies:
algorithms, DB, models, models.crew, models.reactions

Class Hierarchy

- java.lang.Object
  - models.**BaseFilm** (implements DB.ActiveDomainObject)
    - models.**Film** (implements models.IFilm, models.IRateable)
    - models.**Serie** (implements DB.ActiveDomainObject, models.IRateable)
  - models.**Bruker** (implements DB.ActiveDomainObject)
  - models.crew.**CrewMember** (implements DB.ActiveDomainObject)
  - DB.**DBConnection**
  - DB.**DBHelper**
  - models.**Episode** (implements DB.ActiveDomainObject, models.IFilm, models.IRateable)
  - models.**Kategori** (implements DB.ActiveDomainObject)
  - models.reactions.**Kommentar** (implements DB.ActiveDomainObject)
  - **Main**
  - models.**Person** (implements DB.ActiveDomainObject)
  - models.**Produksjonsselskap** (implements DB.ActiveDomainObject)
  - algorithms.**ProfessionalHashingAlgorithm**
  - models.reactions.**Rating** (implements DB.ActiveDomainObject)
  - models.crew.**Skuespiller** (implements DB.ActiveDomainObject)
  - **TextUserInterface**

Interface Hierarchy

- DB.**ActiveDomainObject**
- models.**IFilm**
- models.**IRateable**

Enum Hierarchy

- java.lang.Object
  - java.lang.Enum<E> (implements java.lang.Comparable<T>, java.io.Serializable)
    - models.crew.**CrewTypes**
    - models.**FilmLagetFor**

Figure 1: Overview of the class hierarchy

2.2.1 models.Film, extends models.BaseFilm, implements IFilm and IRateable

Corresponds to a Film in the database - contains the film methods not shared with Series

2.2.2 models.Serie, extends models.BaseFilm, ActiveDomainObject

A class corresponding to a Serie in the database, contains all methods special to Series, not shared with Film through BaseFilm.

2.3 models.Bruker, implements ActiveDomainObject

Represents the user in the database, and is used to login and create a user

2.4 models.crew.CrewMember, implements ActiveDomainObject

Class CrewMember corresponds to all types of crewmembers either in movies or eposodes

2.5 models.crew.CrewMember, implements ActiveDomainObject

Class CrewMember corresponds to all types of crewmembers either in movies or eposodes

2.6 models.Episode, implements ActiveDomainObject, IFilm and IRatable

Represents an episode of a series

2.7 models.Kategori, implements ActiveDomainObject

A class corresponding to Kategori in the database

2.8 models.Kommentar, implements ActiveDomainObject

Class Kommentar corresponds to a comment on a rateable class (movie, series, episode)

2.9 Main

Is used to initialize the connection to the database and create an instance of TextUserInterface

2.10 models.Person, implements ActiveDomainObject

A class corresponding to Person table in the database, contains methods special to person.

2.11 models.Produksjonsselskap, implements ActiveDomainObject

A class corresponding to Produksjonsselskap in db.

2.12 ProfessionalHashingAlgorithm

An instance of this class is used to “hash” the password for users, that are then stored in the database. Not a good hashing algorithm, but could easiliy be replaced.

2.13 models.reactions.rating, implements ActiveDomainObject

A Rating is a rating on a rateable class (movie, series or episode)

2.14 models.crew.Skuespiller, implements ActiveDomainObject

Class Skuespiller corresponds to actors in both movies and episodes

2.15 TextUserInterface

The class used to implement the user interface for the application. This is where all the functionality that the user actually uses is implemented.

3 ii) Overview of how the usecases are realized in the program

The user-interactivity for all usecases are implemented in TextUserInterface. Specific database calls that are not deemed to be useful outside this TextUserInterface class are also implemented here, such as findPersonByLike etc.

3.1 Usecase 1: Find the names of all roles for a given actor

This usecase is implemented in a method called `findAllRolesForActor` in TextUserInterface. This method uses uses another method `findPerson` which reads in the name of the actor and searches the database for an actor with that name and returns it as an Optional<Person>.

If there exists such a person it will call the `findAllActorRoles` method on that Person instance object. It iterates through all the roles and prints them out to stdout.

3.2 Usecase 2: Find all movies a given actor appears in

This is implemented similarly to usecase 1, but in `findAllActorMovieAppearences`. It gets a person with the given name using `findPerson` and calls the `findAllActorRoles` method. Instead of printing out the role it prints out the movie or series name.

3.3 Usecase 3: Find which production companies make the most movies for all genres

This usecase is implemented in `findProductionCompanyWithMostMoviesPerGenre` , which calls `Kategori.findAllCategories()`, iterates through them all and finding all movies for that given category using the Kategori method `findAllFilmsByCategory`. It counts the number of films each production company has produced for the category and stores the production company with the highest amount in a map.

At the end it iterates through the map and prints out the genre and production company that had the most movies for this genre.

3.4 Usecase 4: Insert new movie

Implemented in `insertNewMovie`. Lets the user enter movie details by reading from stdin. For values that are supposed to be in the database, such as directors and production companies, the user first searches for them by name. If they exist in the database then that entry is used, otherwise they are asked to create a new one, and then the newly created one is used.

The user can enter directors, categories, scriptWriters, and actors, along side the details needed for a movie.

3.5 Usecase 5: Insert a new review of an episode of a series

Implemented in `insertNewReview`. First the user is asked to authenticate. This is done by using the `Bruker` class. If the credentials were correct it will continue, otherwise it will go back to the main menu.

The user is prompted to enter a series name to lookup in the database. If none exists, it will return to the main menu. Likewise for finding an episode for the given series.

The `Rating` class is used to create a new rating, and saving it to the database.