CSE488:Ontologies and Semantic Web

Submitted To:

Dr. Ensaf Hussein Mohamed

TA. Eman Khaled

Submitted By:

Ahmed Tarek Abdellatif	20p8417
Mazen Tayseer	20p7460
Mariam Sameh	20p1599
Adham Amr Abdelaty	20p5249

Table of Contents

Github Link :

https://github.com/Adhamamr360/Ontologies-Project

Problem Description:

Problem Domain: Modeling Movie Industry

The movie industry is a vast and dynamic field encompassing various elements such as movies, directors, writers, actors, genres, and production details. Managing and understanding the relationships between these entities is crucial for various stakeholders, including movie enthusiasts, filmmakers, producers, and researchers. However, the complexity of these relationships makes it challenging to organize and analyze movie-related data effectively.

Domain of Interest: Ontology for Movie Representation

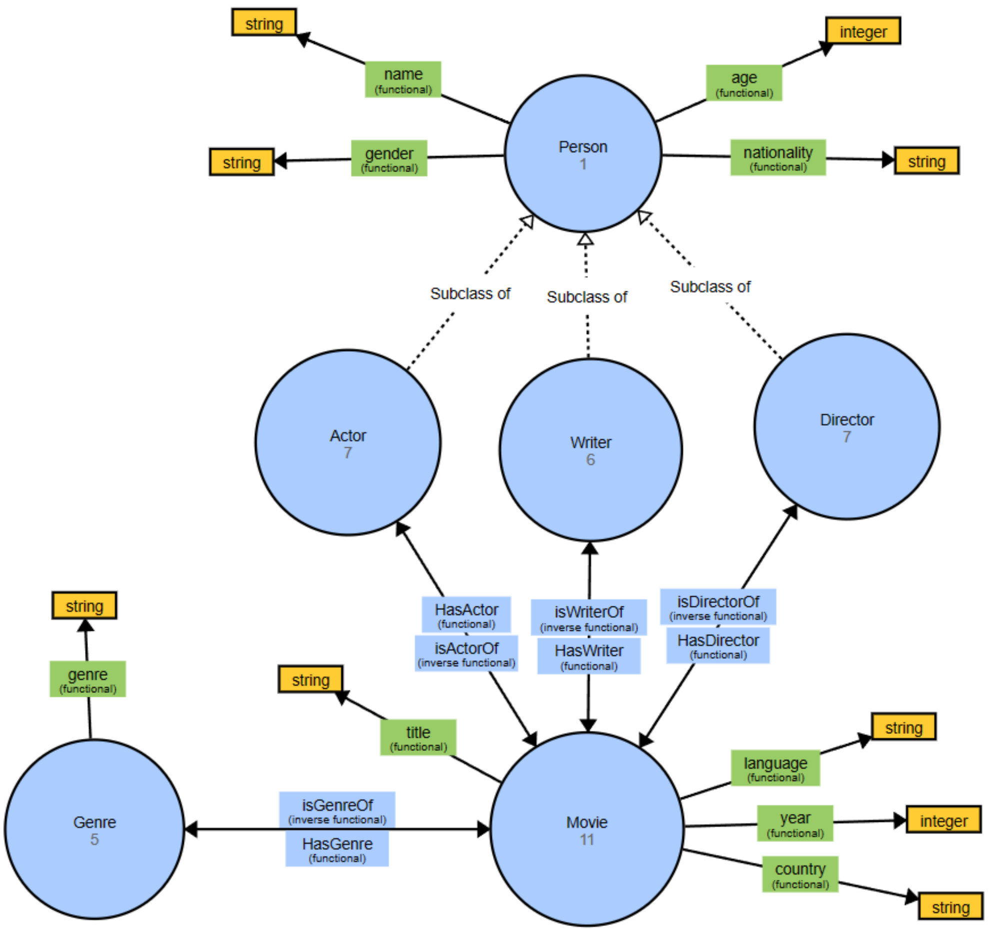
To address the challenges posed by the complexity of the movie industry, an ontology for movie representation is proposed. This ontology aims to provide a structured framework for modeling and organizing movie-related entities and their relationships. By creating a semantic representation of movies, directors, writers, actors, genres, and other relevant information, this ontology facilitates more efficient data management, querying, and reasoning tasks within the movie domain.

Key Objectives:

1. **Modeling Movie Entities**: Define classes and properties to represent movies, directors, writers, actors, genres, and related attributes such as title, year, country, language, etc.
2. **Capturing Relationships**: Specify relationships between entities, such as the association between movies and their directors, writers, and actors, as well as the categorization of movies into different genres.
3. **Ensuring Consistency**: Ensure that the ontology maintains consistency by defining disjoint classes, restrictions, and conditions to enforce valid relationships and data constraints.
4. **Facilitating Querying**: Enable querying of the ontology to retrieve specific information about movies, such as listing all actors or directors, identifying movies of a particular genre, or finding movies written by a specific writer.
5. **Supporting Inference**: Use inference capabilities to derive additional knowledge from the ontology, such as inferring actor-director relationships based on their involvement in movies.
6. **Integration with Applications**: Integrate the ontology with applications and tools to support tasks such as movie recommendation systems, data analytics, and decision-making processes within the movie industry.

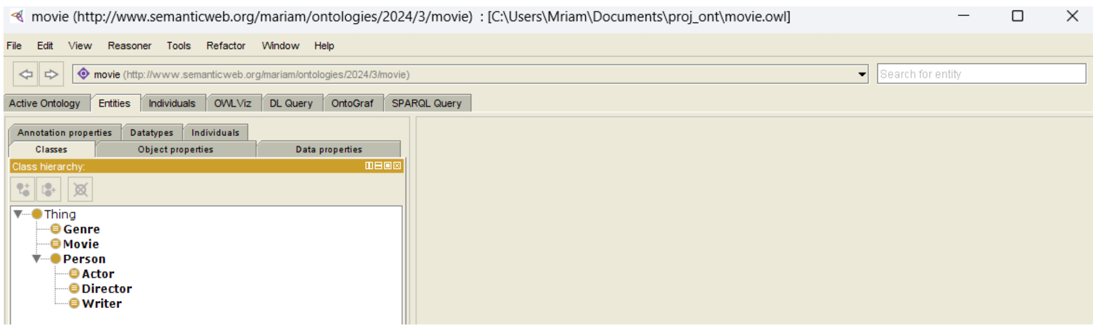
Overall, the ontology for movie representation aims to provide a comprehensive and semantically rich framework for organizing and analyzing movie-related data, thereby enhancing the efficiency and effectiveness of various movie-related tasks and applications.

Part I: Modeling the Ontology



Classes:

1. Actor - represents individuals who play a role in a movie.
2. Director- represents individuals who direct a movie.
3. Writer- represents individuals who write a script for a movie.
4. Person- represents individuals who are involved in the movie industry but not necessarily in a specific role.
5. Movie- represents a film or motion picture.
6. Genre- represents categories or styles of movies, such as action, comedy, drama, etc.


movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]

```
Thing
    Genre
    Movie
    Person
        Actor
        Director
        Writer
```

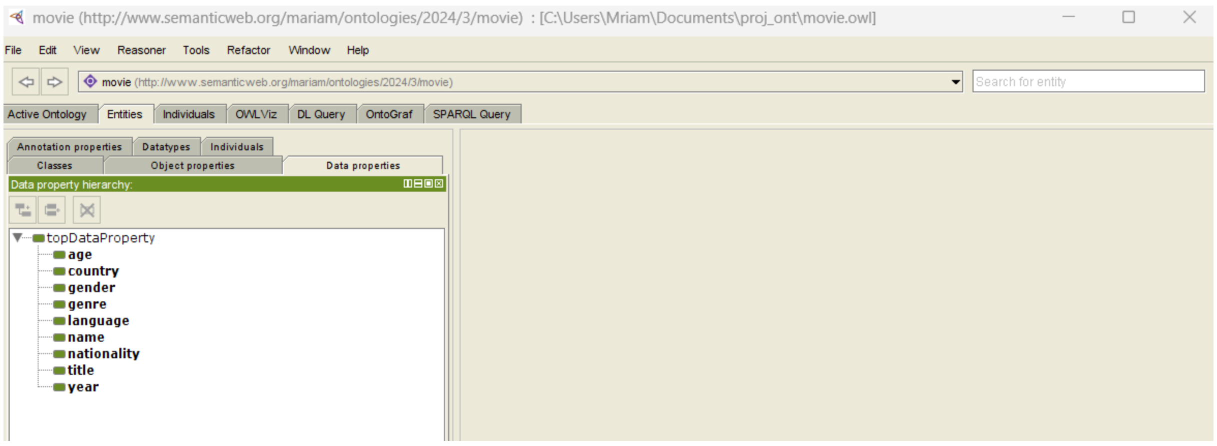
Data Properties:

1. age: domain = Person, range = xsd:integer, characteristic = functional.
2. name: domain = Person, range = xsd:string, characteristic = functional.
3. gender: domain = Person, range = xsd:string, characteristic = functional.
4. nationality: domain = Person, range = xsd:string, characteristic = functional.

Indicates the age, name, gender, and nationality of a person.

5. country: domain = Movie, range = xsd:string, characteristic = functional.
6. language: domain = Movie, range = xsd:string, characteristic = functional.
7. title: domain = Movie, range = xsd:string, characteristic = functional.
8. year: domain = Movie, range = xsd:integer, characteristic = functional.
9. genre: domain = Genre, range = xsd:string, characteristic = functional.

Indicates the country, language, title, genre, and year of release of a movie.

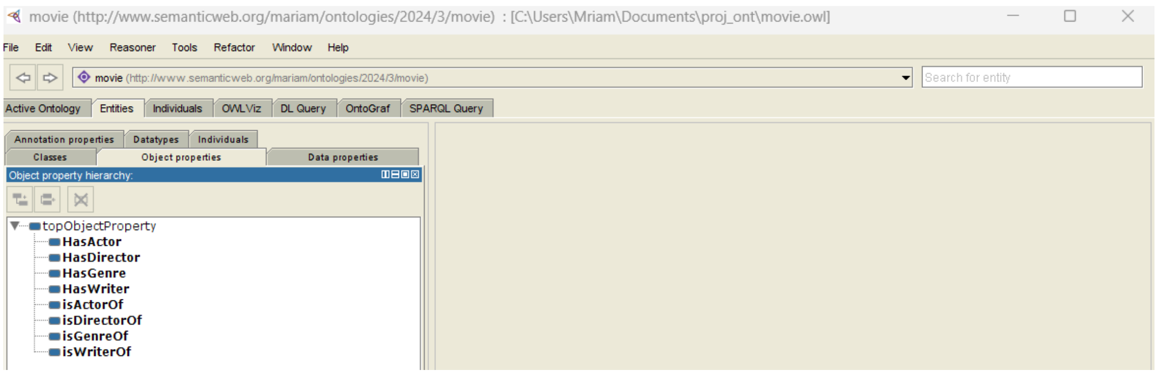


```
topDataProperty
    age
    country
    gender
    genre
    language
    name
    nationality
    title
    year
```

Object Properties:

1. HasActor: domain = Movie, range = Actor, characteristic = functional.
   Indicates that a movie has an actor.
2. HasDirector: domain = Movie, range = Director, characteristic = functional.
   Indicates that a movie has a director
3. HasWriter: domain = Movie, range = Writer, characteristic = functional
   Indicates that a movie has a writer.

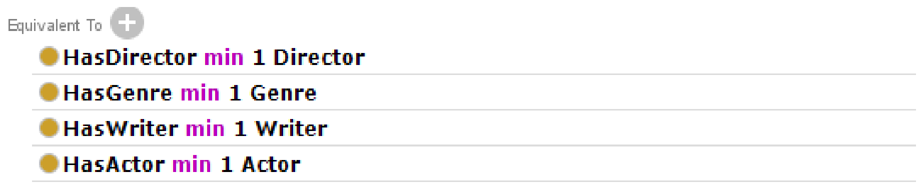
4. isActorOf: domain = Actor, range = Movie, characteristic = inverse functional
   Indicates that an actor has acted in a movie.
5. isDirectorOf: domain = Director, range = Movie, characteristic = functional
   Indicates that a movie is directed by a director.
6. isWriterOf: domain = Writer, range = Movie, characteristic = functional
   Indicates that a movie is written by a writer.
7. HasGenre: domain = Movie, range = Genre, characteristic = functional
   Indicates that a movie has a genre.
8. isGenreOf: domain = Genre, range = Movie, characteristic = inverse functional
   Indicates that a genre belongs to a movie.


movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
Annotation properties Datatypes Individuals
Classes Object properties Data properties
Object property hierarchy:

topObjectProperty
- HasActor
- HasDirector
- HasGenre
- HasWriter
- isActorOf
- isDirectorOf
- isGenreOf
- isWriterOf

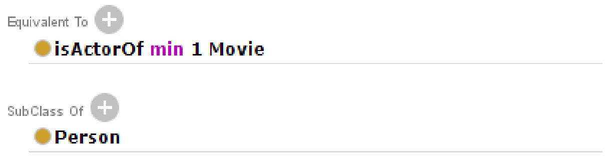
Constraints:

1. Class Movie, Genre, and Person are disjoint classes.
2. Class Actor, Movie, and Genre are disjoint classes.
3. Class Director, Movie, and Genre are disjoint classes.
4. Class Writer, Movie, and Genre are disjoint classes.
5. A movie must have at least one actor (min 1 HasActor).
6. A movie must have at least one director (min 1 HasDirector).
7. A movie must have at least one writer (min 1 HasWriter).


Equivalent To

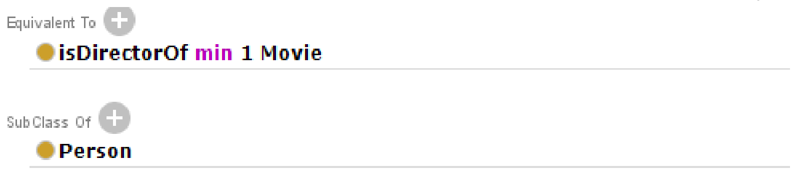
HasDirector min 1 Director
HasGenre min 1 Genre
HasWriter min 1 Writer
HasActor min 1 Actor

8. An actor must have acted in at least one movie (min 1 isActorOf).


Equivalent To
isActorOf min 1 Movie

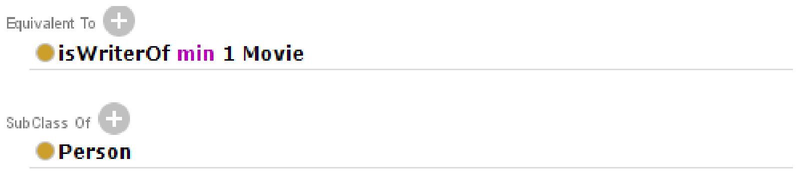
SubClass Of
Person

9. A director must have directed at least one movie (min 1 isDirectorOf).


Equivalent To
isDirectorOf min 1 Movie

SubClass Of
Person

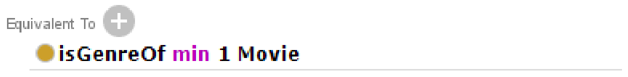
10. A writer must have written for at least one movie (min 1 isWriterOf).


Equivalent To
isWriterOf min 1 Movie

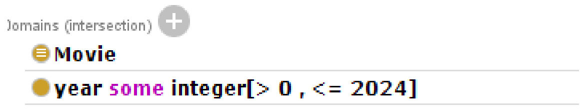
SubClass Of
Person

11. A movie must have at least one genre (min 1 genre).

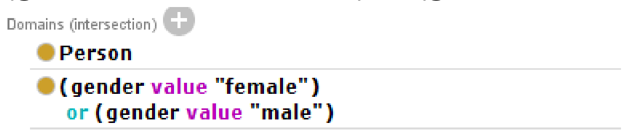
12. A genre must have at least one movie (min 1 movie)


Equivalent To
isGenreOf min 1 Movie

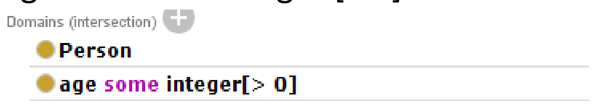
13. year some xsd:integer [>0<= 2024]


)omains (intersection)
Movie
year some integer[> 0 , <= 2024]

14. (gender value "female") or (gender value "male")


Domains (intersection)
Person
(gender value "female")
or (gender value "male")

15. age some xsd:integer [> 0]


Domains (intersection)
Person
age some integer[> 0]

Part II: Populating the Ontology

Individuals:

1. Some individuals for **Genre** class:
   a) Thriller, a genre of movie.
   b) Crime, a genre of movie.
   c) Action, a genre of movie.
   d) Comedy, a genre of movie.
   e) Drama, a genre of movie.

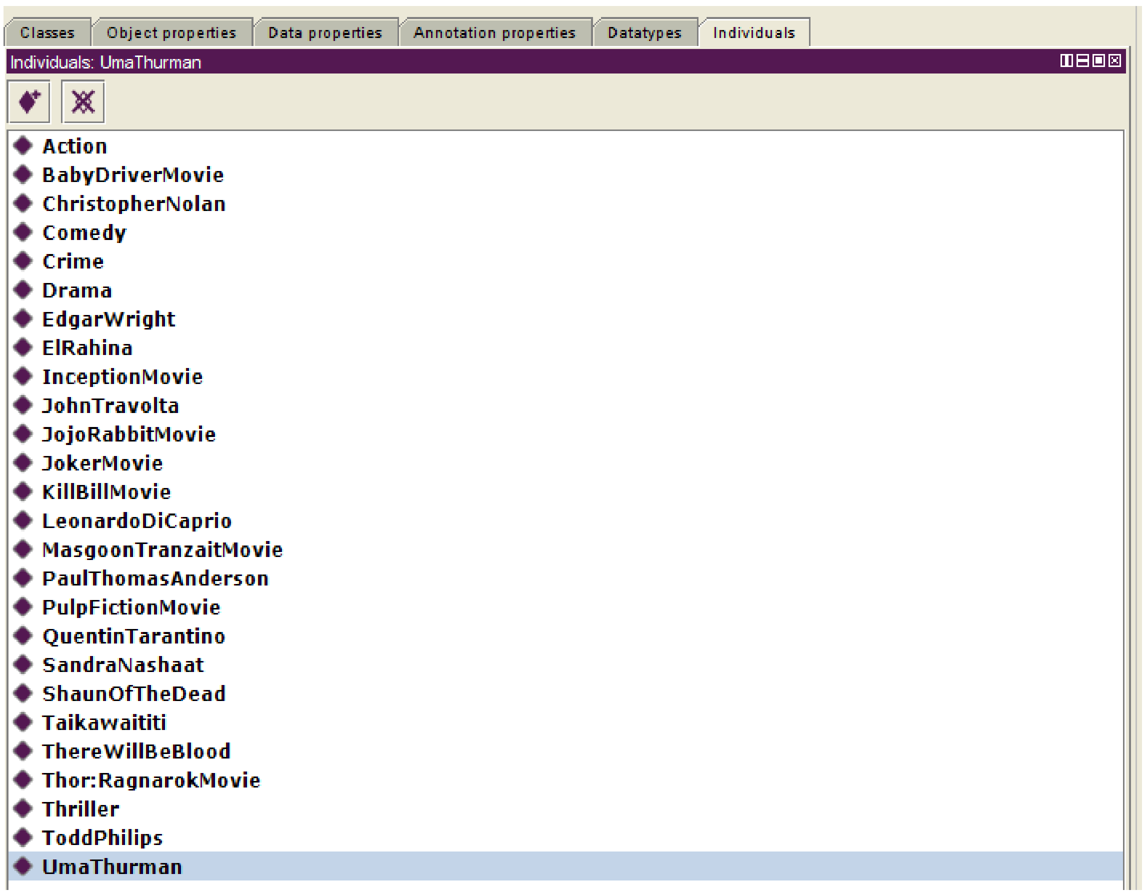
2. Some individuals to the **Movie** class:
   a) Pulp Fiction, Genre: Crime Thriller, 1994, USA, English.
   b) Kill Bill (volume 1), Genre: Action Crime Thriller, 2003, USA, English.
   c) Inception, Genre = Action Drama, 2010, USA, English.
   d) Joker, Genre = Crime Drama, 2019, USA, English.
   e) Baby Driver, Genre=Drama Comedy Action,2017,UK,English.
   f) Jojo Rabbit, Genre= Drama Comedy,2019,USA,English.
   g) Masgoon Tranzait,Genre=Action,2008,Egypt,Arabic.
   h) Shaun of the Dead, Genre=Comedy Thriller,2004,USA,English.
   i) There will be Blood, Genre= Drama Thriller,2007,USA,English.
   j) Thor:Rangnarok, Genre=Action Drama,2017,USA,English.
   k) El Rahina, Genre=Comedy Action,2006,Egypt,Arabic.
3. Some individuals to the **Person** class:
   a) Quentin Tarantino, American, 53 years old, writer and director of Pulp Fiction and Kill Bill(volume1). He also played a role in that movie.
   b) John Travolta, American, 59 years old, actor in Pulp Fiction.
   c) Uma Thurman, American, 43 years old, actress in Pulp Fiction. She also participated as awriter in Kill Bill (volume1).
   d) Christopher Nolan, British-American, 52 years old, writer and director of Inception.
   e) Leonardo DiCaprio, American, 47 years old, actor in Inception.
   f) Edgar Wright,British,48 years old,Director and writer of baby driver and shaun of the dead and also he played a role in shaun of the dead.
   g) Taika Waititi, a New Zealand director, 47 years old, known for his work on "Thor: Ragnarok" and "Jojo Rabbit". Waititi not only

directs but also frequently appears in his own films, showcasing his versatility as both a filmmaker and actor.

h) Paul Thomas Anderson, an American filmmaker, aged 51, known for writing and directing "There Will Be Blood" He often includes himself in small acting roles within his films.

i) Sandra Nashaat, Female Egyptian Director,54 years old, director of elRahina and Masgoon Tranzait.

j) Todd Philips , American , 51 years old, director and writer of the Joker Movie.

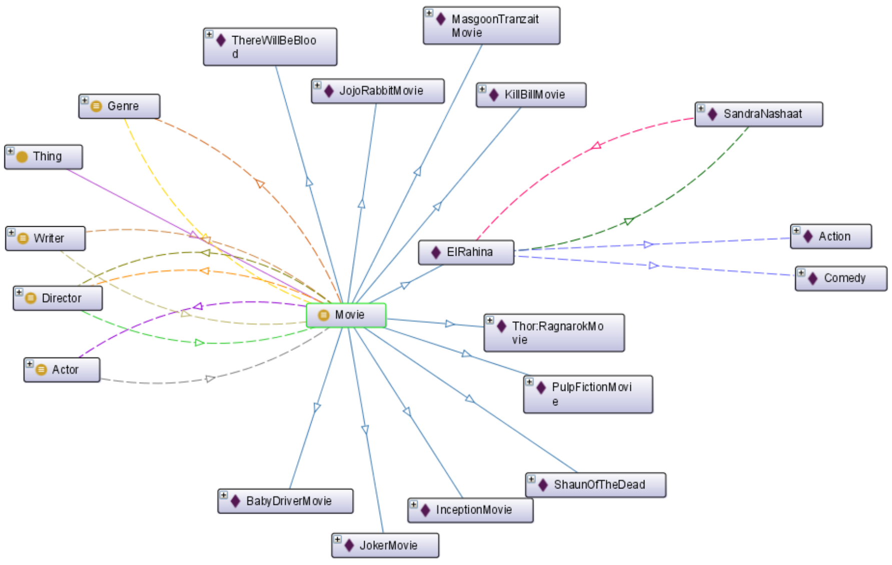


- Action
- BabyDriverMovie
- ChristopherNolan
- Comedy
- Crime
- Drama
- EdgarWright
- ElRahina
- InceptionMovie
- JohnTravolta
- JojoRabbitMovie
- JokerMovie
- KillBillMovie
- LeonardoDiCaprio
- MasgoonTranzaitMovie
- PaulThomasAnderson
- PulpFictionMovie
- QuentinTarantino
- SandraNashaat
- ShaunOfTheDead
- Taikawaititi
- ThereWillBeBlood
- Thor:RagnarokMovie
- Thriller
- ToddPhilips
- UmaThurman

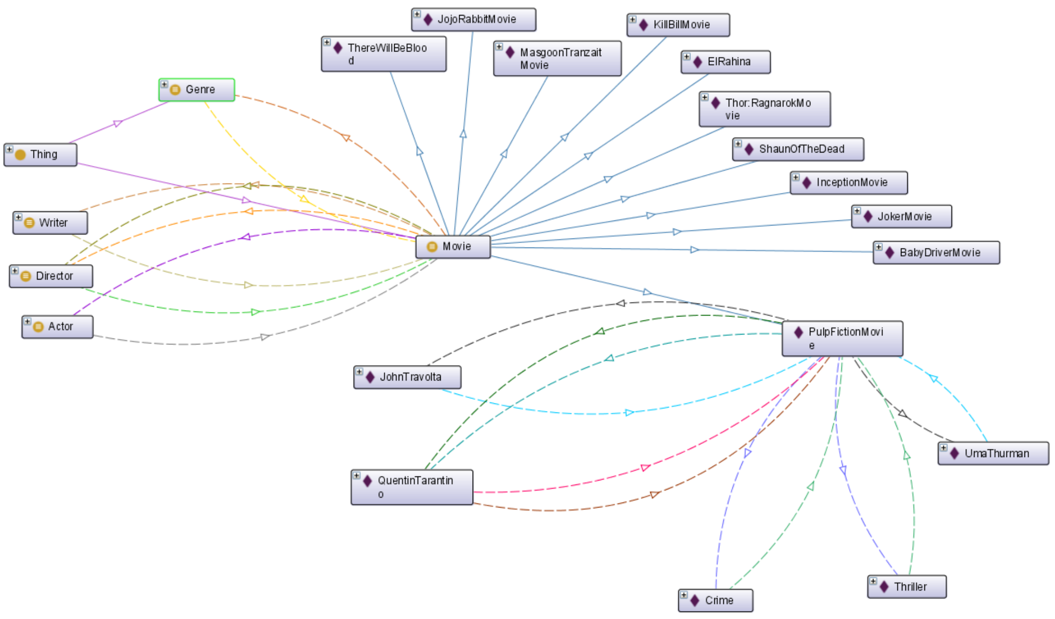
Ontology Graph:

Movie Instances

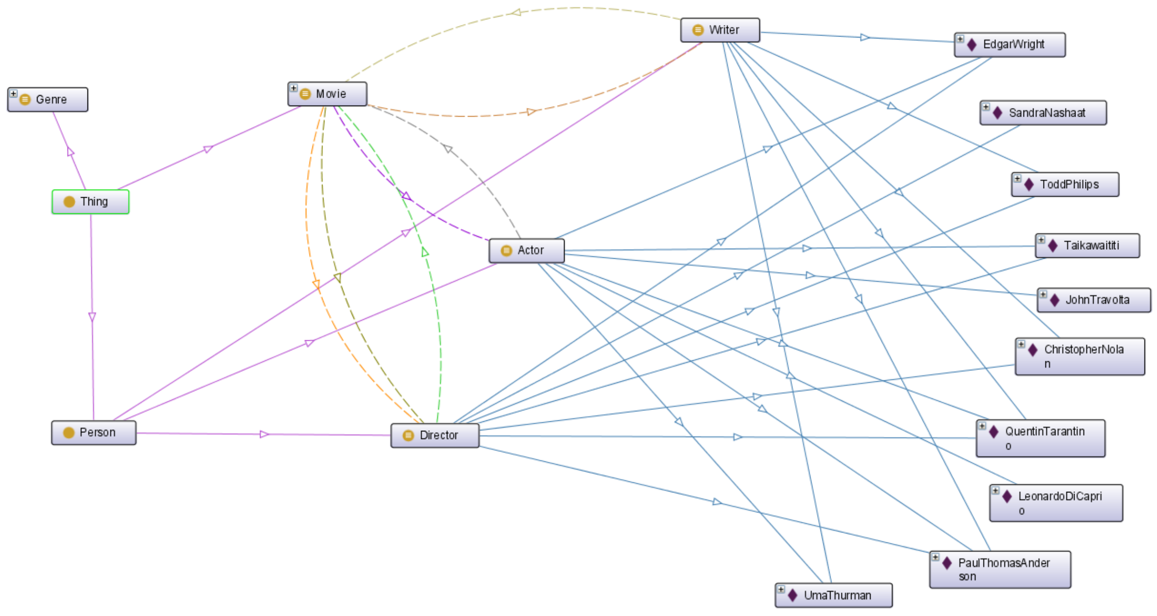
- Example **El Rahina** Movie



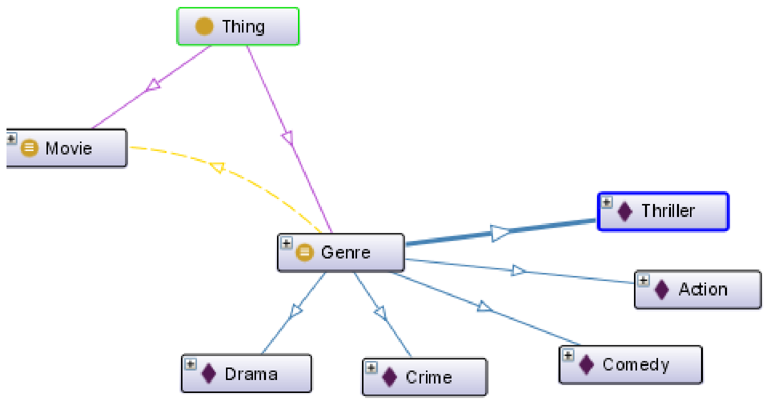
- Example **Pulp Fiction** Movie



Person Instances



Genre Instances

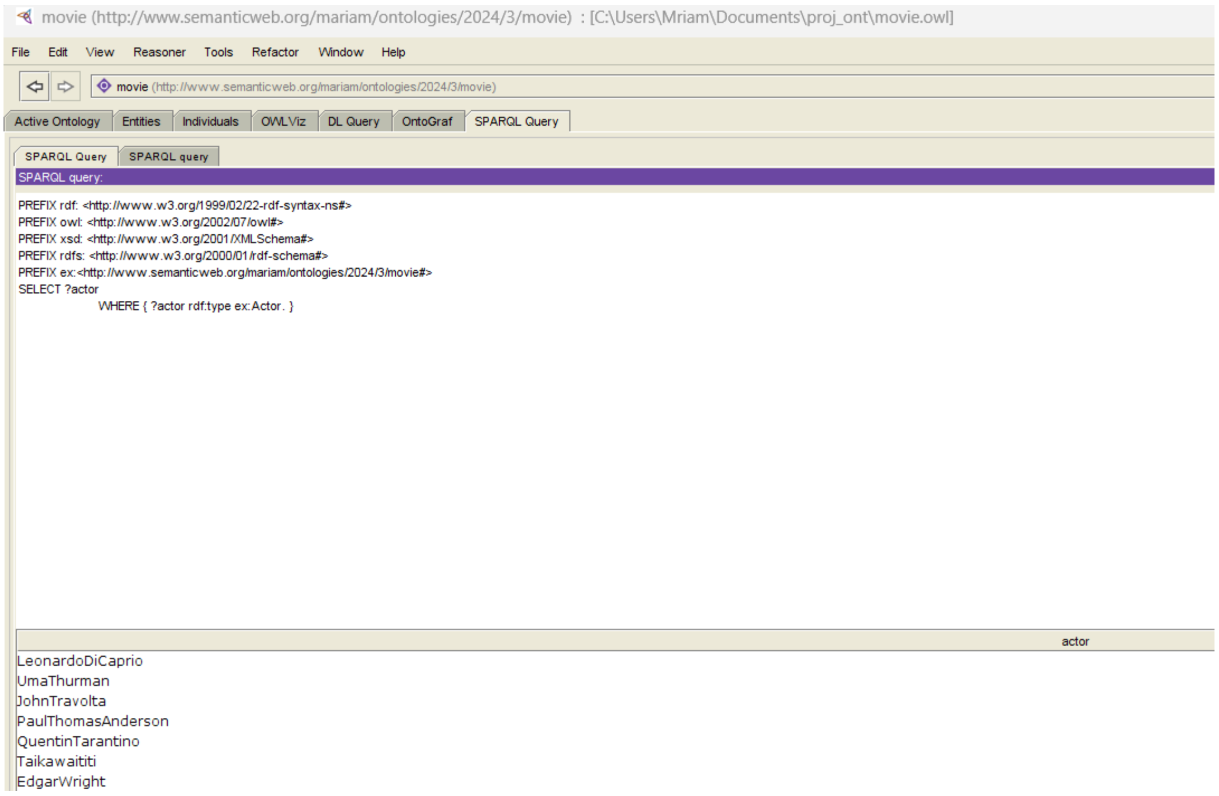


Part III: Querying the Ontology

Write SPARQL queries to response to the following:

1. **List the instances of the class Actor**

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?actor

        WHERE { ?actor rdf:type ex:Actor. }
```



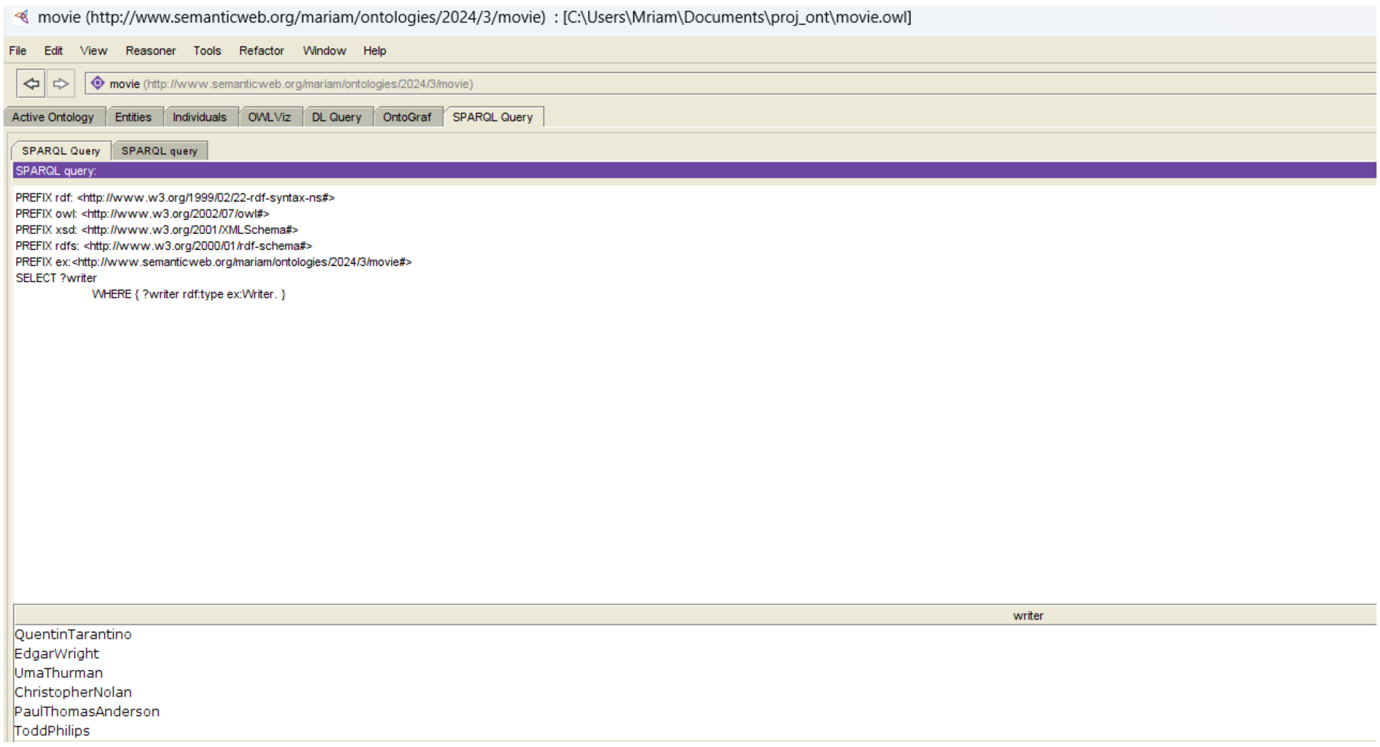
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
SPARQL Query SPARQL query
SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?actor
        WHERE { ?actor rdf:type ex:Actor. }
```

```
actor
LeonardoDiCaprio
UmaThurman
JohnTravolta
PaulThomasAnderson
QuentinTarantino
Taikawaititi
EdgarWright
```

2. List the instances of the class writer

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?writer

        WHERE { ? writer rdf:type ex: Writer. }
```


movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
SPARQL Query SPARQL query
SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?writer
            WHERE { ?writer rdf:type ex:Writer. }
```

```
writer
QuentinTarantino
EdgarWright
UmaThurman
ChristopherNolan
PaulThomasAnderson
ToddPhilips
```

3. List the instances of the class director

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

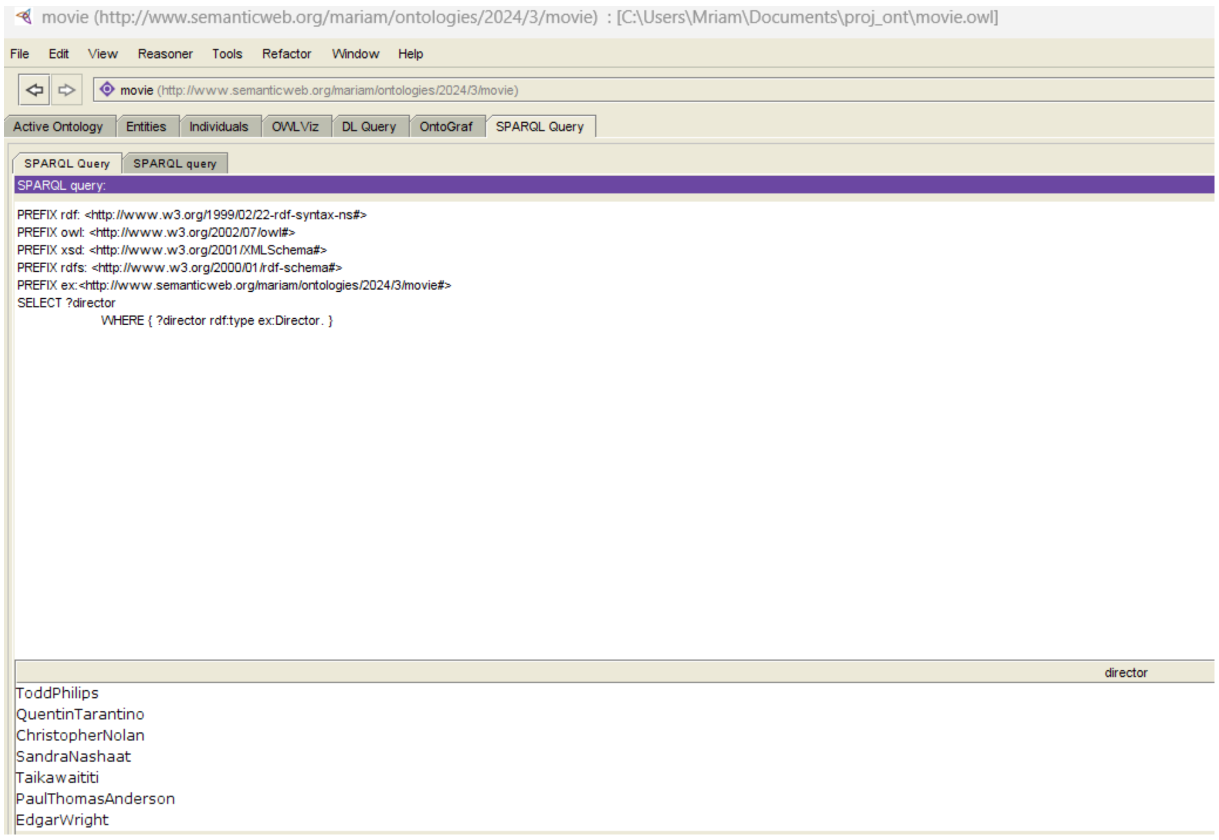
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?director

WHERE { ? director rdf:type ex: Director. }

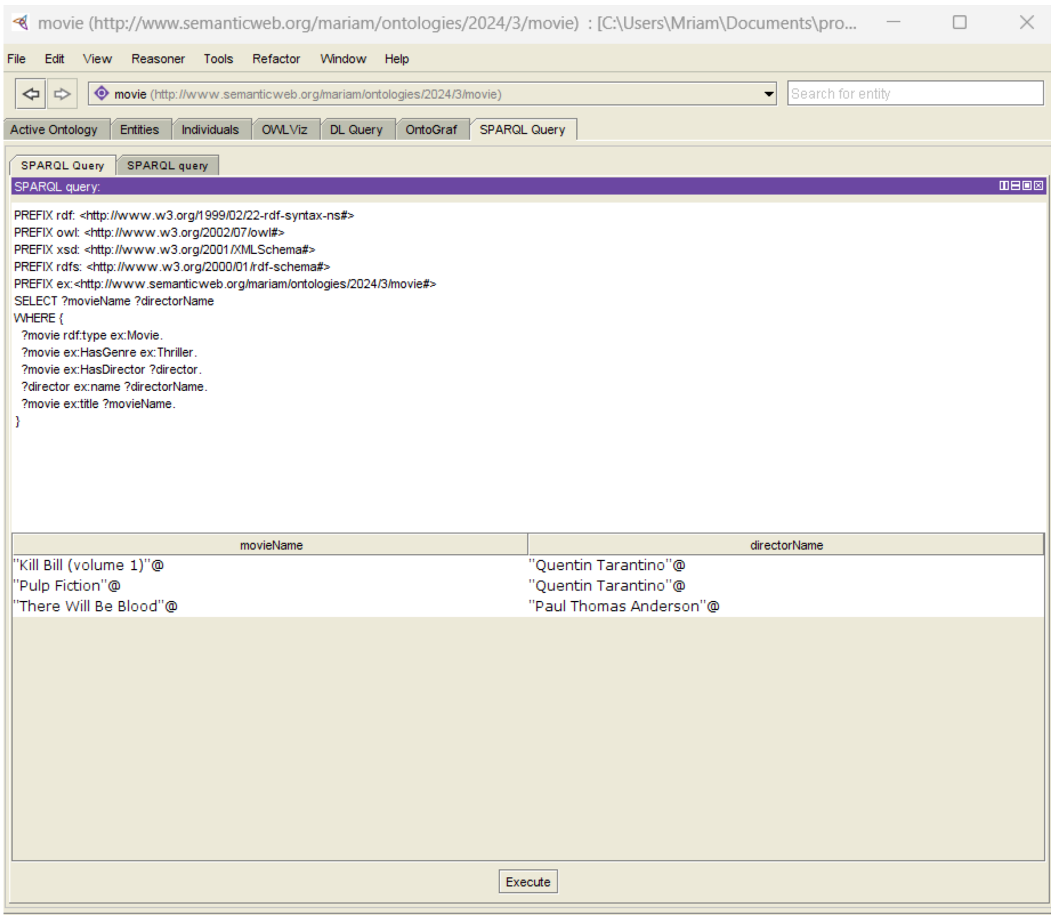

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]
File
Edit
View
Reasoner
Tools
Refactor
Window
Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Active Ontology
Entities
Individuals
OWLViz
DL Query
OntoGraf
SPARQL Query
SPARQL Query
SPARQL query
SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?director
        WHERE { ?director rdf:type ex:Director. }
```

```
director
ToddPhilips
QuentinTarantino
ChristopherNolan
SandraNashaat
Taikawaititi
PaulThomasAnderson
EdgarWright
```

4. List the name of all Thriller movies. For each one, display its director

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?movieName ?directorName
WHERE {
  ?movie rdf:type ex:Movie.
  ?movie ex:HasGenre ex:Thriller.
  ?movie ex:HasDirector ?director.
  ?director ex:name ?directorName.
  ?movie ex:title ?movieName.
}
```

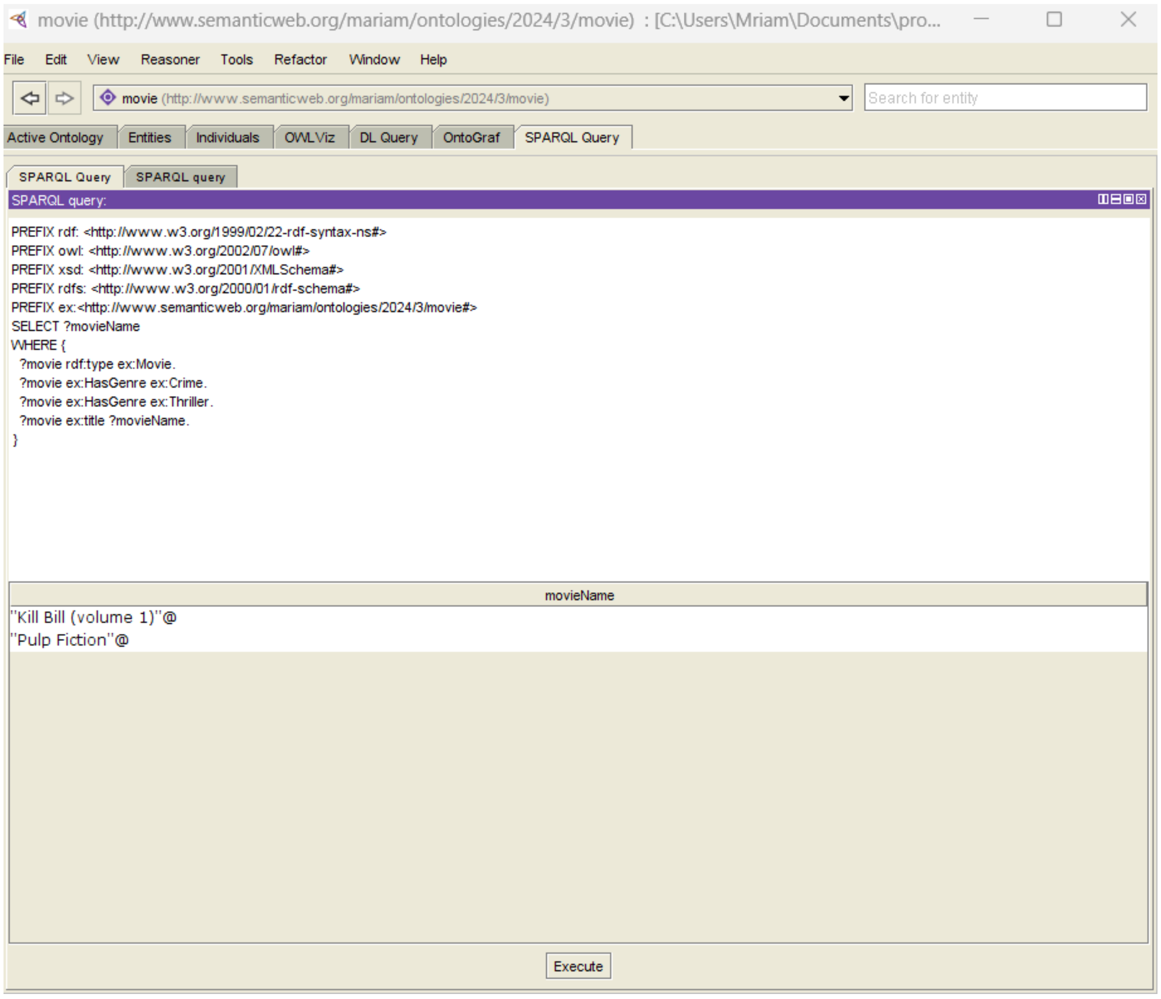

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
SPARQL Query SPARQL query
SPARQL query:
Execute

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?movieName ?directorName
WHERE {
  ?movie rdf:type ex:Movie.
  ?movie ex:HasGenre ex:Thriller.
  ?movie ex:HasDirector ?director.
  ?director ex:name ?directorName.
  ?movie ex:title ?movieName.
}
```

movieName	directorName
"Kill Bill (volume 1)"@	"Quentin Tarantino"@
"Pulp Fiction"@	"Quentin Tarantino"@
"There Will Be Blood"@	"Paul Thomas Anderson"@

5. List the name of all Crime Thriller movies

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?movieName

WHERE {

  ?movie rdf:type ex:Movie.

  ?movie ex:HasGenre ex:Crime.

  ?movie ex:HasGenre ex:Thriller.

  ?movie ex:title ?movieName.

}
```

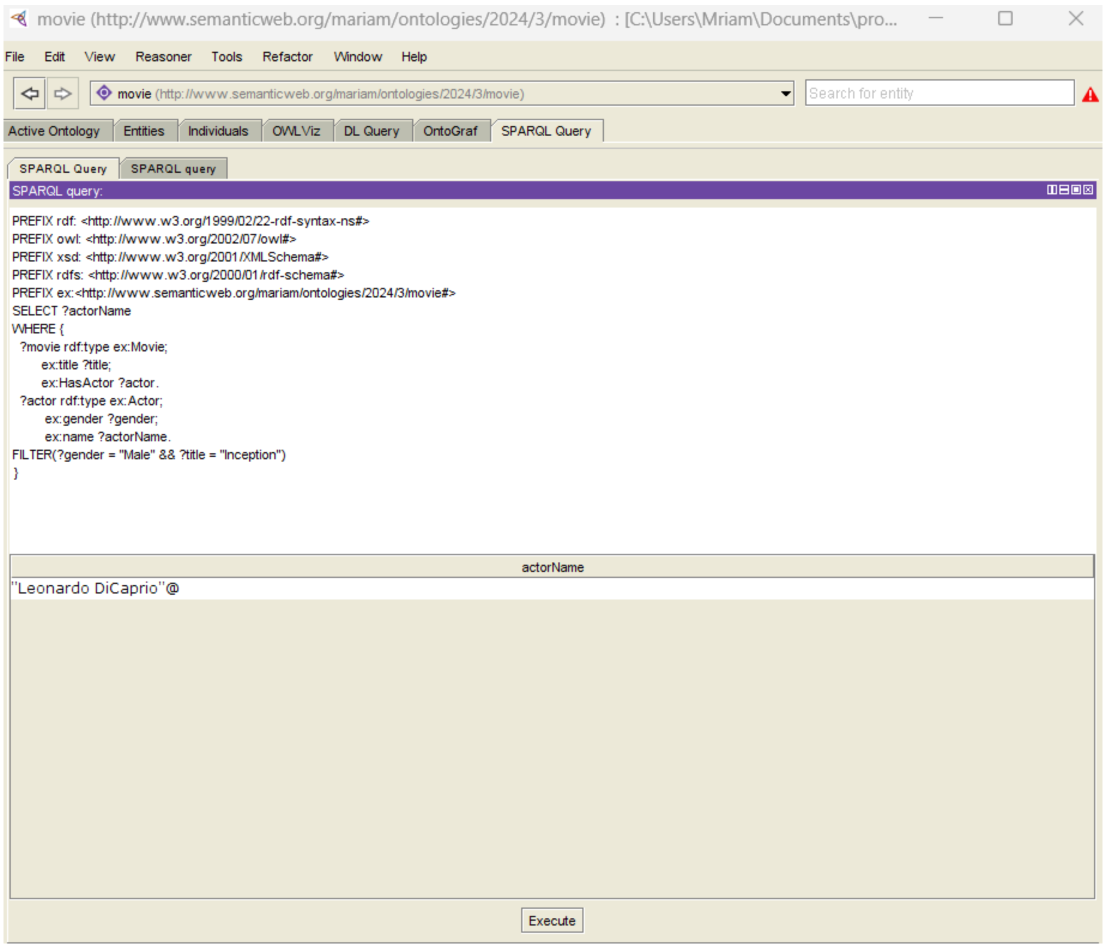

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
SPARQL Query SPARQL query
SPARQL query:
Execute

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?movieName
WHERE {
  ?movie rdf:type ex:Movie.
  ?movie ex:HasGenre ex:Crime.
  ?movie ex:HasGenre ex:Thriller.
  ?movie ex:title ?movieName.
}
```

movieName
"Kill Bill (volume 1)"@
"Pulp Fiction"@

6. list the male actors in the movie in specific film

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?actorName
WHERE {
  ?movie rdf:type ex:Movie;
       ex:title ?title;
       ex:HasActor ?actor.
  ?actor rdf:type ex:Actor;
        ex:gender ?gender;
        ex:name ?actorName.
FILTER(?gender = "Male" && ?title = "Inception")
}
```

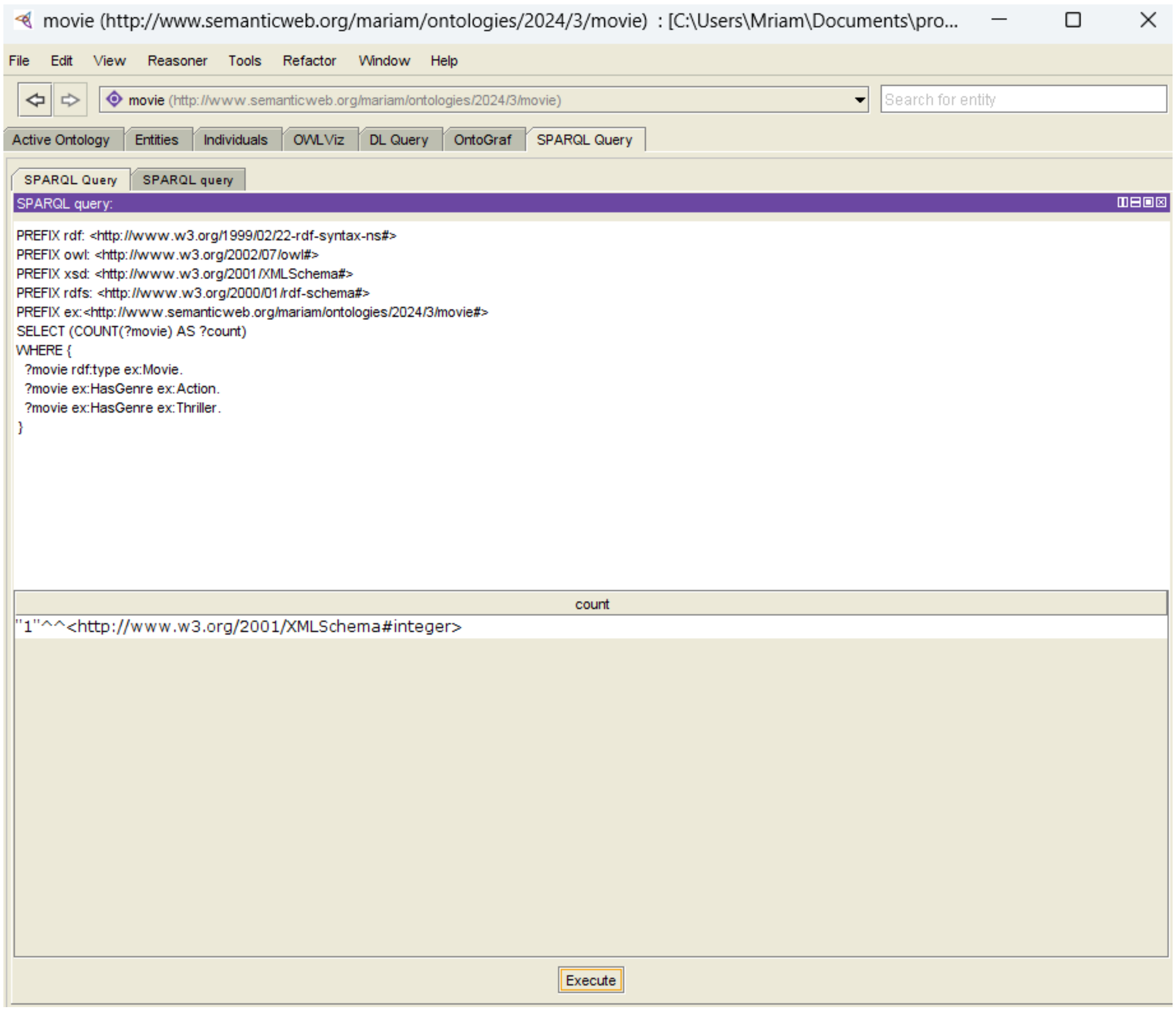

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
SPARQL Query SPARQL query
SPARQL query:
Execute

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?actorName
WHERE {
  ?movie rdf:type ex:Movie;
       ex:title ?title;
       ex:HasActor ?actor.
  ?actor rdf:type ex:Actor;
        ex:gender ?gender;
        ex:name ?actorName.
FILTER(?gender = "Male" && ?title = "Inception")
}
```

```
actorName
"Leonardo DiCaprio"@
```

7. How many movies have both "Action" and "Thriller" as genres?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT (COUNT(?movie) AS ?count)

WHERE {

  ?movie rdf:type ex:Movie.

  ?movie ex:HasGenre ex:Action.

  ?movie ex:HasGenre ex:Thriller.

}
```

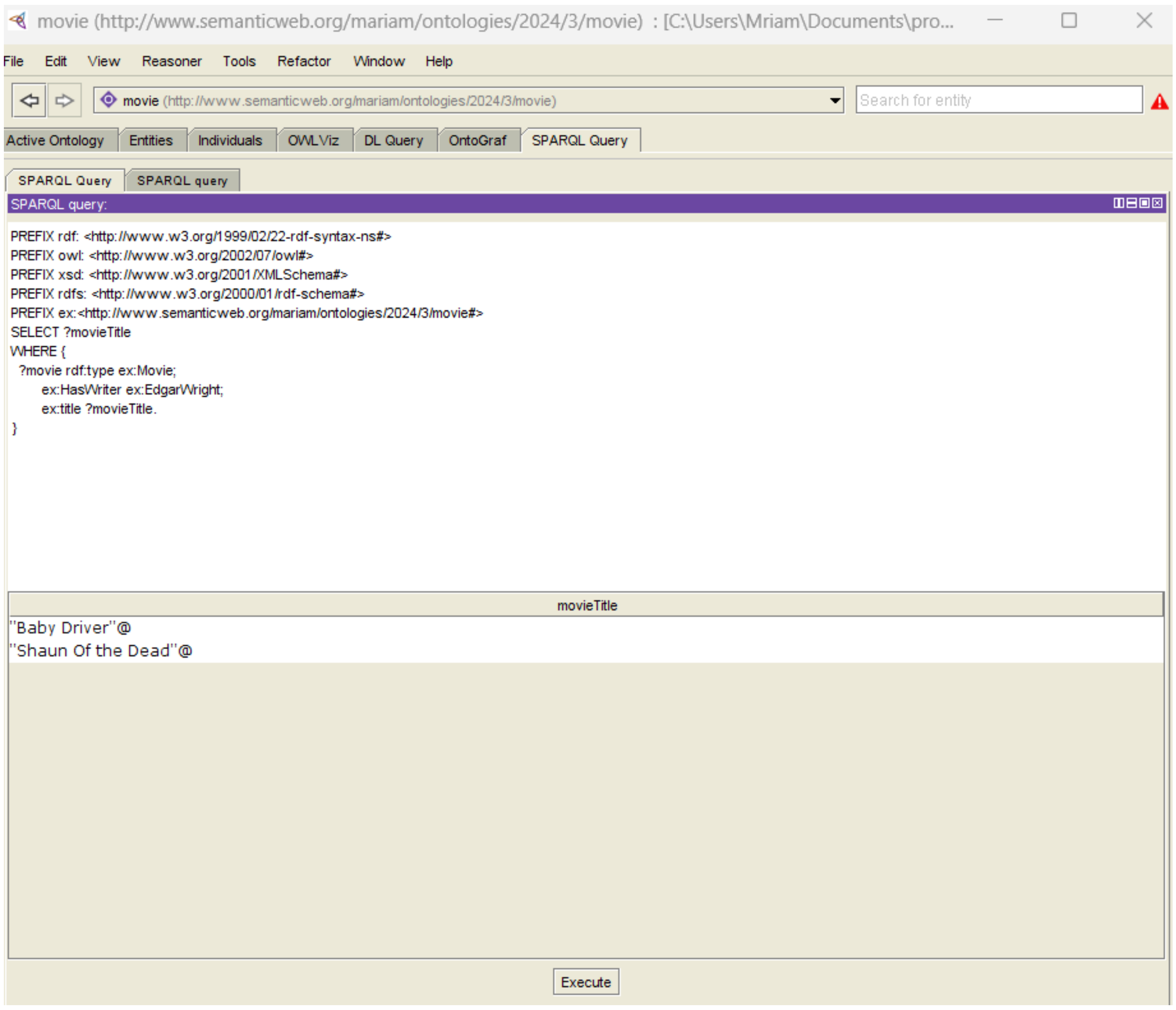

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
SPARQL Query SPARQL query
SPARQL query:
Execute

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT (COUNT(?movie) AS ?count)
WHERE {
 ?movie rdf:type ex:Movie.
 ?movie ex:HasGenre ex:Action.
 ?movie ex:HasGenre ex:Thriller.
}
```

count
"1"^^<http://www.w3.org/2001/XMLSchema#integer>

8. List all the movies written by a specific writer

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?movieTitle

WHERE {

  ?movie rdf:type ex:Movie;

      ex:HasWriter ex:EdgarWright;

      ex:title ?movieTitle.

}
```


movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?movieTitle
WHERE {
 ?movie rdf:type ex:Movie;
      ex:HasWriter ex:EdgarWright;
      ex:title ?movieTitle.
}
```

```
movieTitle
"Baby Driver"@
"Shaun Of the Dead"@
```

9. Find movies with a certain language.

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

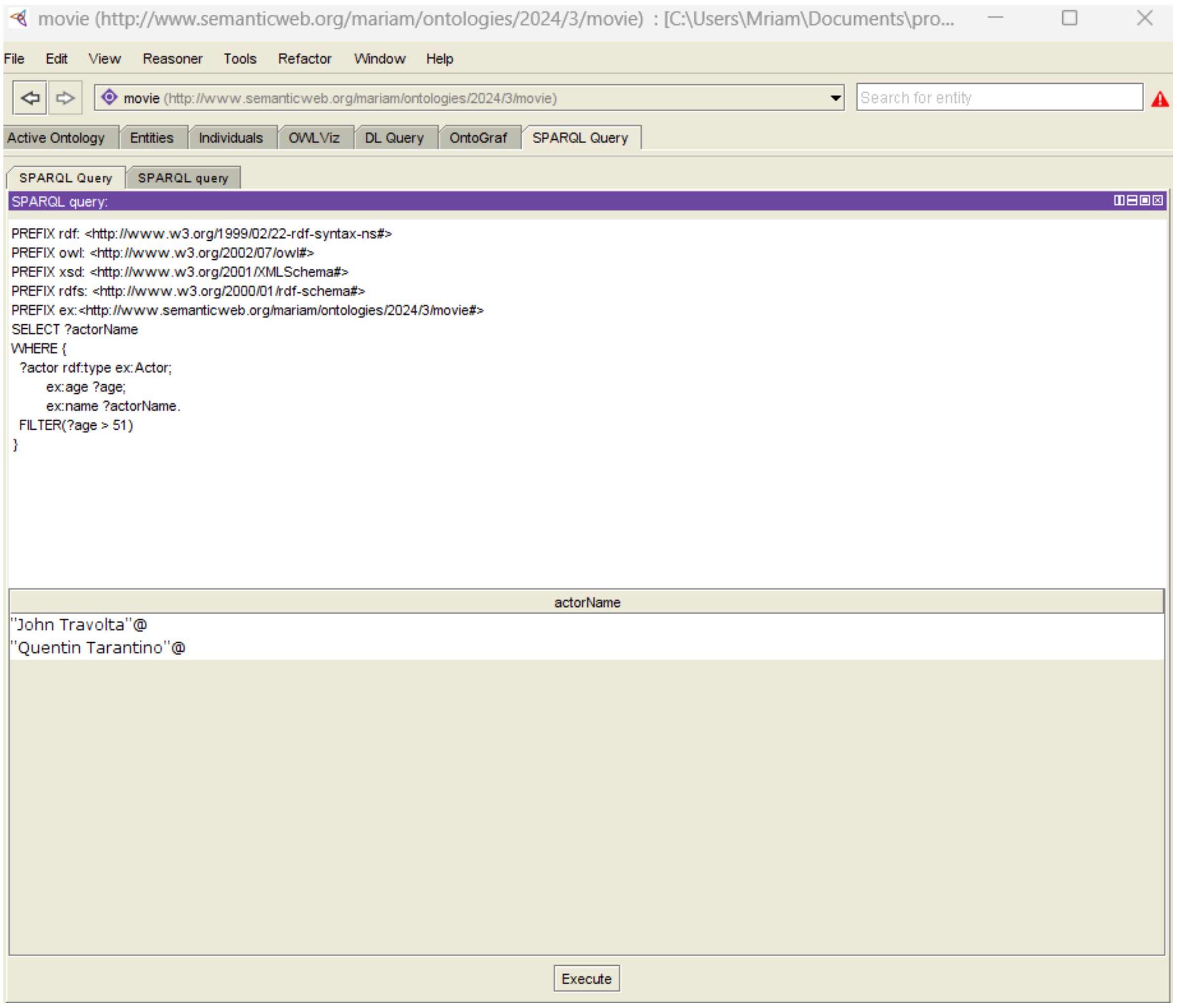
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?actorName

WHERE {

```
 ?actor rdf:type ex:Actor;
        ex:age ?age;
        ex:name ?actorName.
 FILTER(?age > 51)
}
```

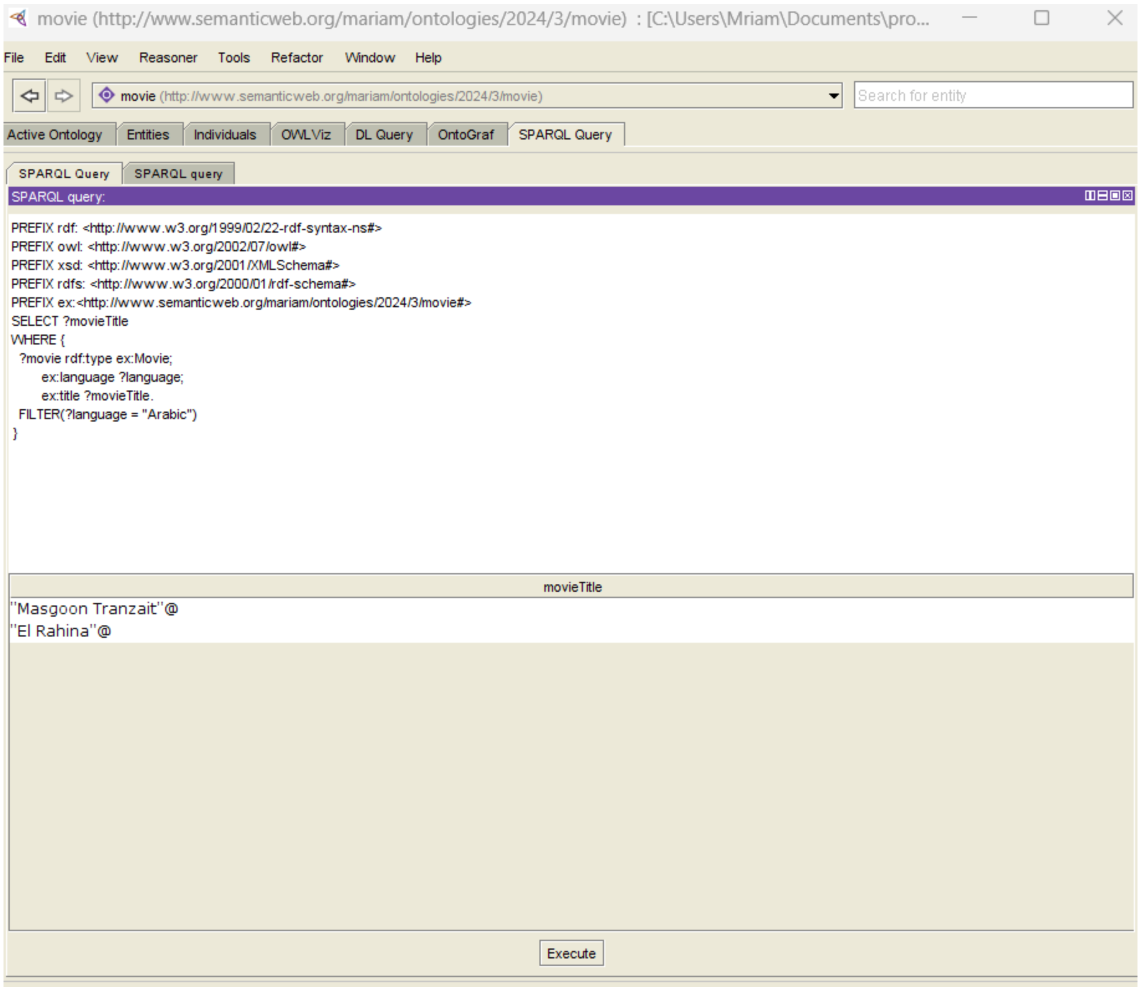

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
SPARQL Query SPARQL query
SPARQL query:
Execute

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?actorName
WHERE {
 ?actor rdf:type ex:Actor;
       ex:age ?age;
       ex:name ?actorName.
 FILTER(?age > 51)
}
```

```
actorName
"John Travolta"@
"Quentin Tarantino"@
```

10. List the name of Actors older than 51 years.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?movieTitle

WHERE {

 ?movie rdf:type ex:Movie;

      ex:language ?language;

      ex:title ?movieTitle.

 FILTER(?language = "Arabic")

}
```


movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
SPARQL Query SPARQL query
SPARQL query:
Execute

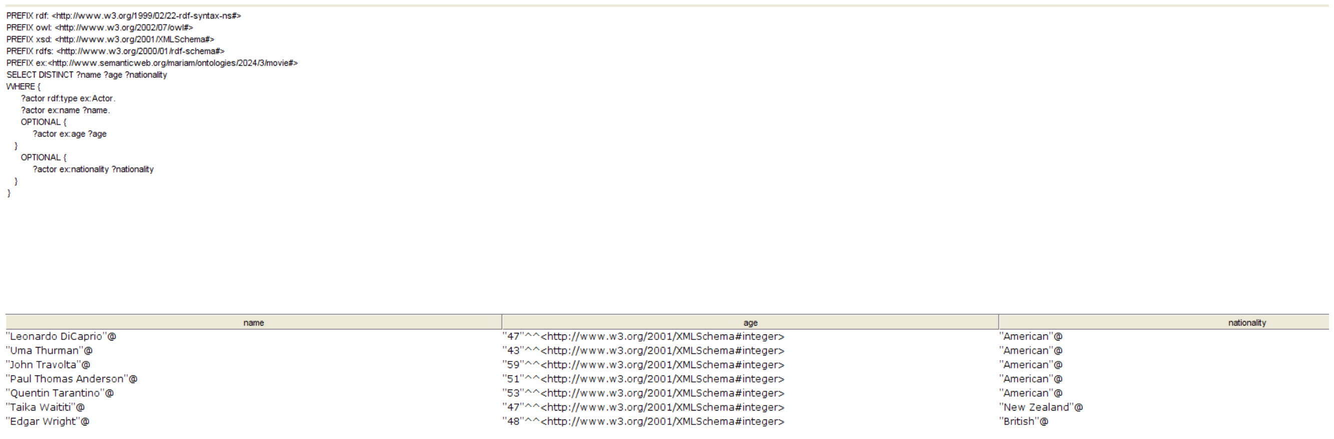
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?movieTitle
WHERE {
 ?movie rdf:type ex:Movie;
      ex:language ?language;
      ex:title ?movieTitle.
 FILTER(?language = "Arabic")
}
```

movieTitle
"Masgoon Tranzait"@
"El Rahina"@

Propose 10 SPARQL queries:

1. **In this example, the query selects actors and includes optional graph patterns to retrieve age and nationality if available. The OPTIONAL keyword is used to specify the optional graph patterns.**

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT DISTINCT ?name ?age ?nationality

WHERE {

    ?actor rdf:type ex:Actor.

    ?actor ex:name ?name.

    OPTIONAL {

        ?actor ex:age ?age

  }

    OPTIONAL {

        ?actor ex:nationality ?nationality

  }

}
```

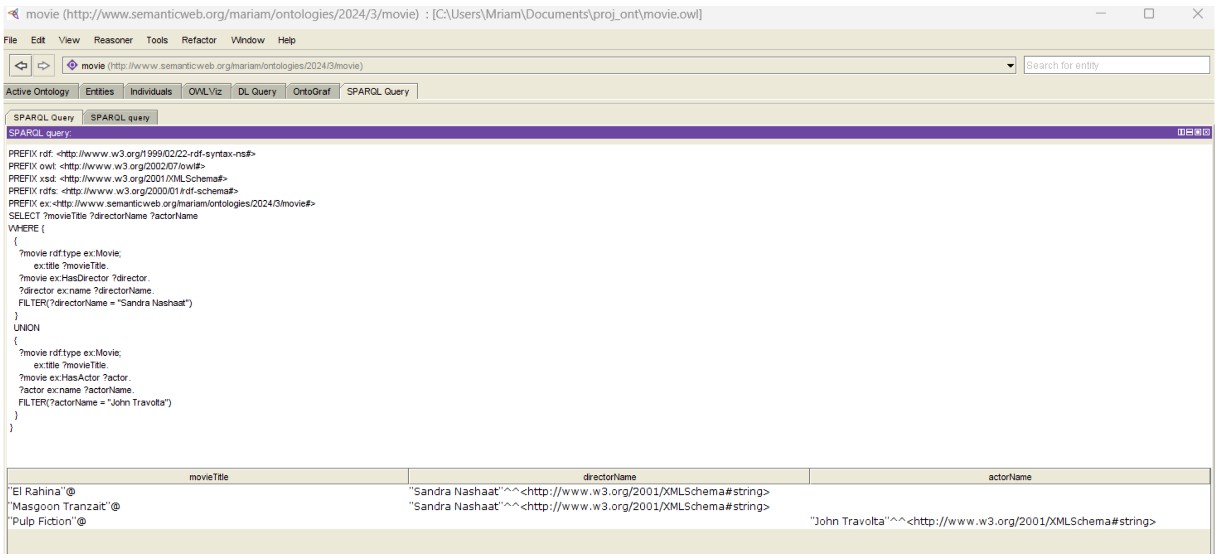


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT DISTINCT ?name ?age ?nationality
WHERE {
    ?actor rdf:type ex:Actor.
    ?actor ex:name ?name.
    OPTIONAL {
        ?actor ex:age ?age
  }
    OPTIONAL {
        ?actor ex:nationality ?nationality
  }
}
```

name	age	nationality
"Leonardo DiCaprio"@	"47"^^<http://www.w3.org/2001/XMLSchema#integer>	"American"@
"Uma Thurman"@	"43"^^<http://www.w3.org/2001/XMLSchema#integer>	"American"@
"John Travolta"@	"59"^^<http://www.w3.org/2001/XMLSchema#integer>	"American"@
"Paul Thomas Anderson"@	"51"^^<http://www.w3.org/2001/XMLSchema#integer>	"American"@
"Quentin Tarantino"@	"53"^^<http://www.w3.org/2001/XMLSchema#integer>	"American"@
"Taika Waititi"@	"47"^^<http://www.w3.org/2001/XMLSchema#integer>	"New Zealand"@
"Edgar Wright"@	"48"^^<http://www.w3.org/2001/XMLSchema#integer>	"British"@

2. **This SPARQL query retrieves information about movies, including their titles, directors, and actors. It employs two alternatives (using UNION) and conjunctions (using FILTER) to combine patterns for extracting data related to directors and actors.**

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?movieTitle ?directorName ?actorName
WHERE {
 {  ?movie rdf:type ex:Movie;
       ex:title ?movieTitle.
   ?movie ex:HasDirector ?director.
   ?director ex:name ?directorName.
   FILTER(?directorName = "Sandra Nashaat")}
 UNION
  { ?movie rdf:type ex:Movie;
       ex:title ?movieTitle.
   ?movie ex:HasActor ?actor.
   ?actor ex:name ?actorName.
   FILTER(?actorName = "John Travolta")}
}
```

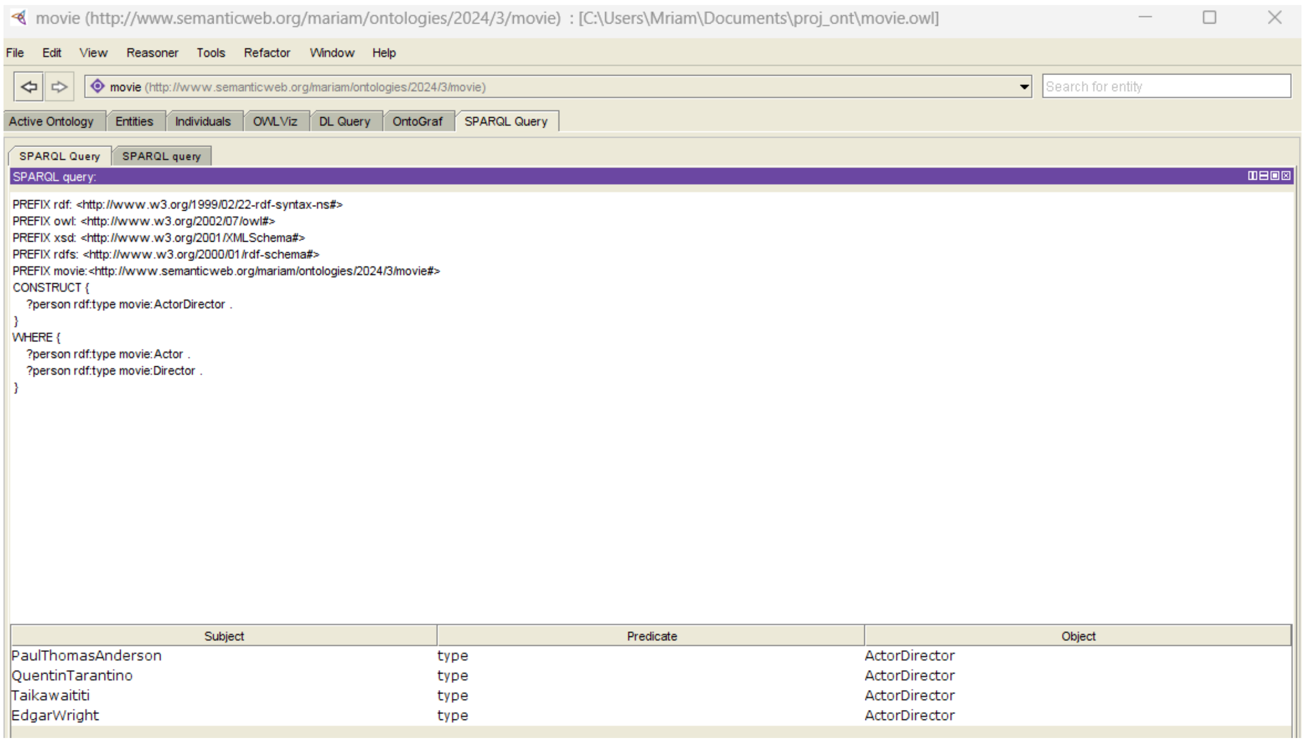

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology Entities Individuals OWLViz DL Query OntoGraf SPARQL Query
SPARQL Query SPARQL query
SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
SELECT ?movieTitle ?directorName ?actorName
WHERE {
 {
  ?movie rdf:type ex:Movie;
     ex:title ?movieTitle.
  ?movie ex:HasDirector ?director.
  ?director ex:name ?directorName.
  FILTER(?directorName = "Sandra Nashaat")
 }
 UNION
 {
  ?movie rdf:type ex:Movie;
     ex:title ?movieTitle.
  ?movie ex:HasActor ?actor.
  ?actor ex:name ?actorName.
  FILTER(?actorName = "John Travolta")
 }
}
```

movieTitle	directorName	actorName
"El Rahina"@	"Sandra Nashaat"^^<http://www.w3.org/2001/XMLSchema#string>	
"Masgoon Tranzait"@	"Sandra Nashaat"^^<http://www.w3.org/2001/XMLSchema#string>	
"Pulp Fiction"@		"John Travolta"^^<http://www.w3.org/2001/XMLSchema#string>

3. CONSTRUCT a new RDF graph that includes individuals who are both actors and directors.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX movie:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

CONSTRUCT {

   ?person rdf:type movie:ActorDirector .

}

WHERE {

   ?person rdf:type movie:Actor .

   ?person rdf:type movie:Director .

}
```


movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology
Entities
Individuals
OWLViz
DL Query
OntoGraf
SPARQL Query
SPARQL Query
SPARQL query
SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX movie:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
CONSTRUCT {
   ?person rdf:type movie:ActorDirector .
}
WHERE {
   ?person rdf:type movie:Actor .
   ?person rdf:type movie:Director .
}
```

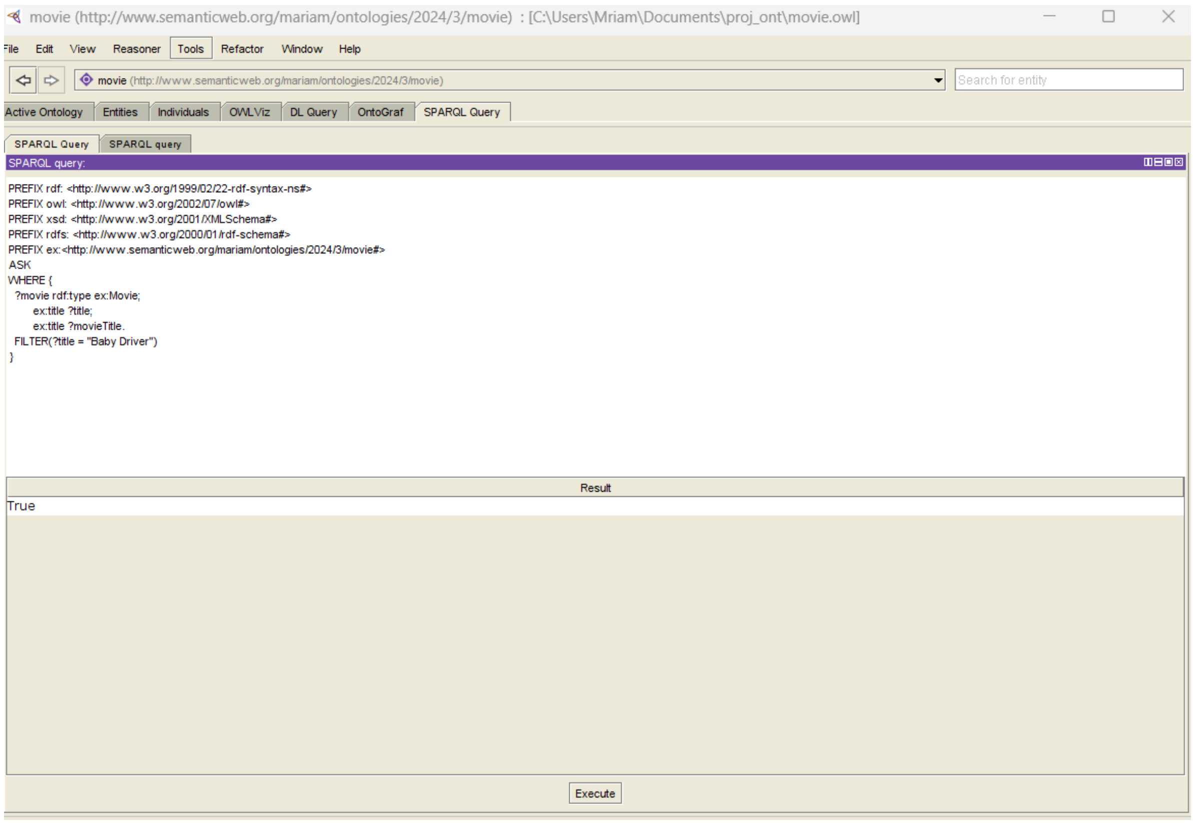
Subject	Predicate	Object
PaulThomasAnderson	type	ActorDirector
QuentinTarantino	type	ActorDirector
Taikawaititi	type	ActorDirector
EdgarWright	type	ActorDirector

4. In this example, the ASK query form is used to check if there are any resources that match the specified patterns in the WHERE clause.

The WHERE clause contains patterns that describe the conditions that need to be satisfied for the query to return a boolean result. In this case, it checks if there is a resource of type "Movie" with the title "Baby Driver".

When you execute this query, the result will be a boolean value indicating whether there are any resources that match the specified patterns. If there are matches, the result will be true; otherwise, it will be false.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

ASK

WHERE {

 ?movie rdf:type ex:Movie;

     ex:title ?title;

     ex:title ?movieTitle.

 FILTER(?title = "Baby Driver")

}
```


movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]
File
Edit
View
Reasoner
Tools
Refactor
Window
Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology
Entities
Individuals
OWLViz
DL Query
OntoGraf
SPARQL Query
SPARQL Query
SPARQL query
SPARQL query:
Result
Execute

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
ASK
WHERE {
 ?movie rdf:type ex:Movie;
     ex:title ?title;
     ex:title ?movieTitle.
 FILTER(?title = "Baby Driver")
}
```

```
True
```

5. **In this example, the DESCRIBE query form is used to retrieve information about a specific resource (Director). The DESCRIBE query returns a description of the specified resource, including its properties and connected resources.**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

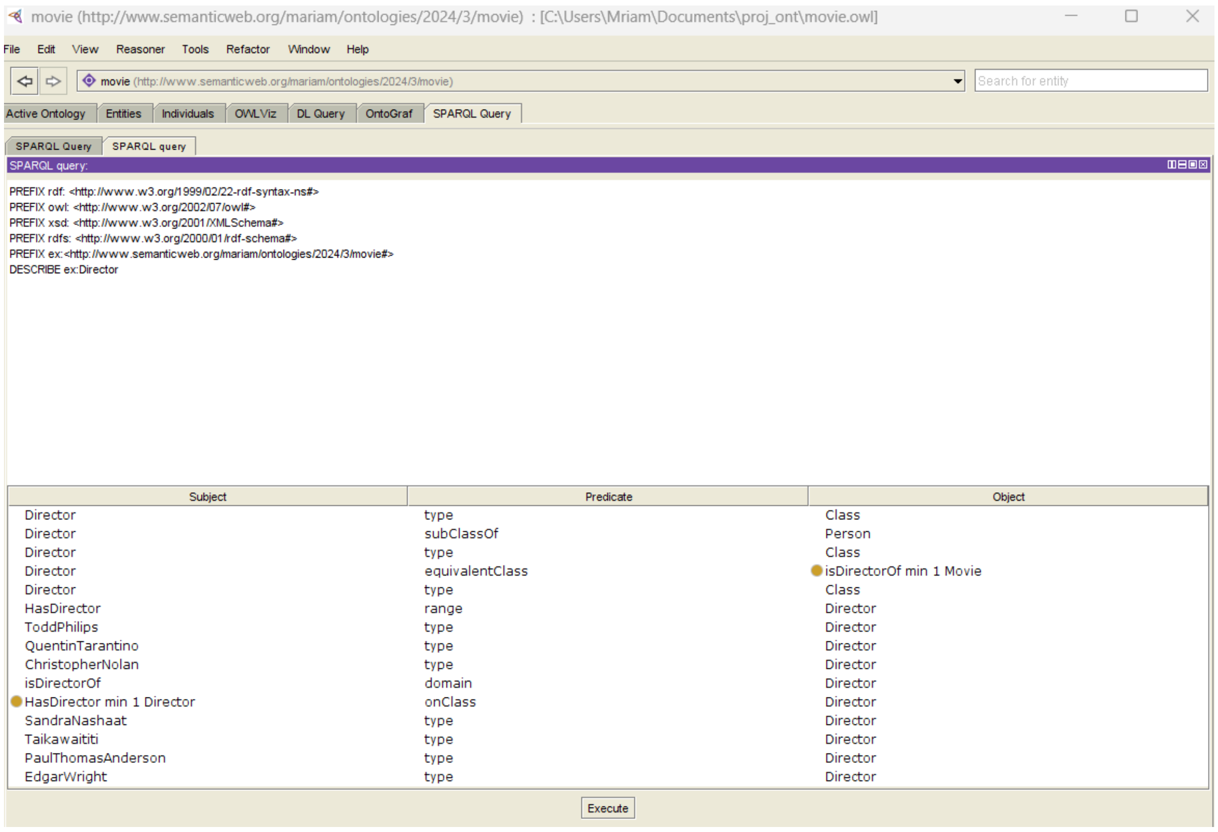
PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

DESCRIBE ex:Director

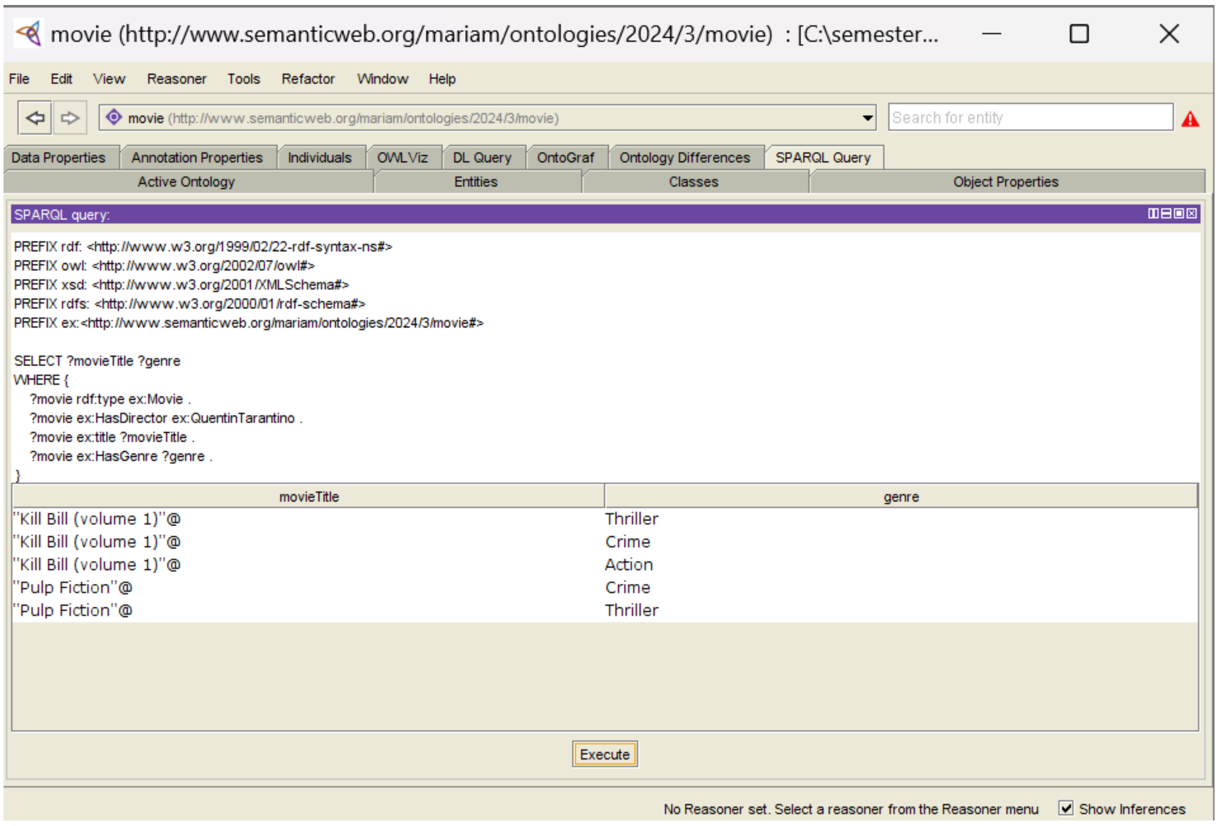

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Active Ontology
Entities
Individuals
OWLViz
DL Query
OntoGraf
SPARQL Query
SPARQL Query
SPARQL query
SPARQL query:
Execute

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
DESCRIBE ex:Director
```

Subject	Predicate	Object
Director	type	Class
Director	subClassOf	Person
Director	type	Class
Director	equivalentClass	● isDirectorOf min 1 Movie
Director	type	Class
HasDirector	range	Director
ToddPhilips	type	Director
QuentinTarantino	type	Director
ChristopherNolan	type	Director
isDirectorOf	domain	Director
● HasDirector min 1 Director	onClass	Director
SandraNashaat	type	Director
Taikawaititi	type	Director
PaulThomasAnderson	type	Director
EdgarWright	type	Director

6. This query provides a simple way to retrieve information about movies directed by Quentin Tarantino, including their titles and genres..

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>


SELECT ?movieTitle ?genre

WHERE {

   ?movie rdf:type ex:Movie .

   ?movie ex:HasDirector ex:QuentinTarantino .

   ?movie ex:title ?movieTitle .

   ?movie ex:HasGenre ?genre .

}
```

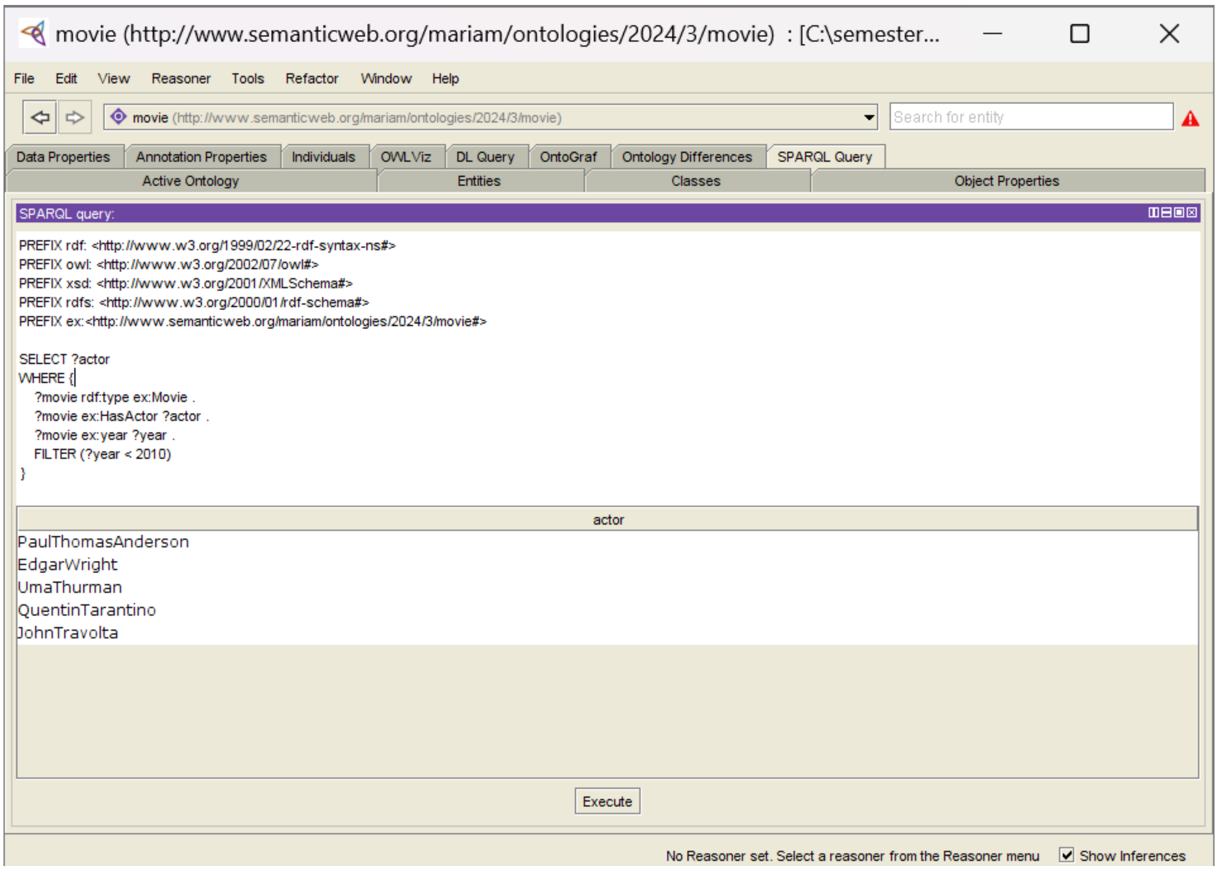

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\semester...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Active Ontology Entities Classes Object Properties
SPARQL query:
Execute
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?movieTitle ?genre
WHERE {
   ?movie rdf:type ex:Movie .
   ?movie ex:HasDirector ex:QuentinTarantino .
   ?movie ex:title ?movieTitle .
   ?movie ex:HasGenre ?genre .
}
```

```
movieTitle                    genre
"Kill Bill (volume 1)"@       Thriller
"Kill Bill (volume 1)"@       Crime
"Kill Bill (volume 1)"@       Action
"Pulp Fiction"@               Crime
"Pulp Fiction"@               Thriller
```

7. In this example, We List all actors who have appeared in movies released before 2010.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>


SELECT ?actor

WHERE {

  ?movie rdf:type ex:Movie .

  ?movie ex:HasActor ?actor .

  ?movie ex:year ?year .

  FILTER (?year < 2010)

}
```

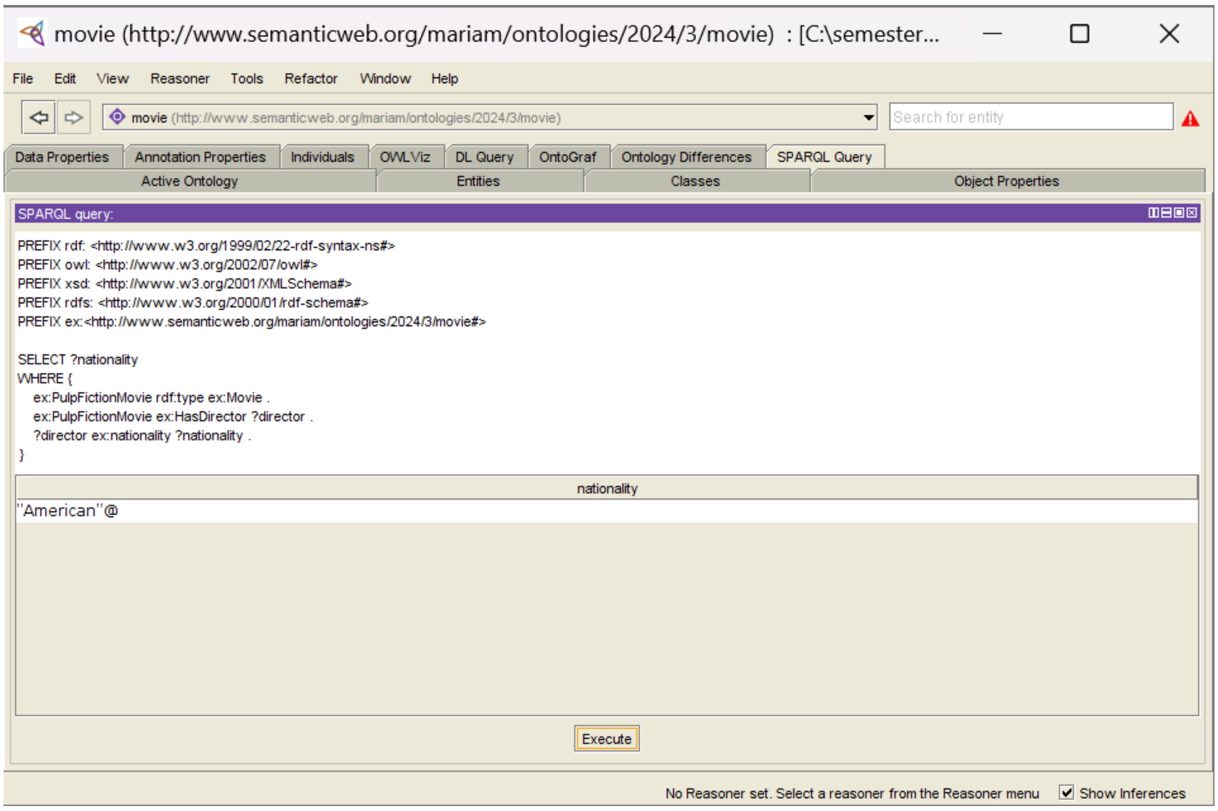

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\semester...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Active Ontology Entities Classes Object Properties
SPARQL query:
Execute
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?actor
WHERE {
  ?movie rdf:type ex:Movie .
  ?movie ex:HasActor ?actor .
  ?movie ex:year ?year .
  FILTER (?year < 2010)
}
```

actor
PaulThomasAnderson
EdgarWright
UmaThurman
QuentinTarantino
JohnTravolta

8. In this example, We Find the nationality of the director of a specific movie (e.g., Pulp Fiction).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>


SELECT ?nationality

WHERE {

   ex:PulpFictionMovie rdf:type ex:Movie .

   ex:PulpFictionMovie ex:HasDirector ?director .

   ?director ex:nationality ?nationality .

}
```

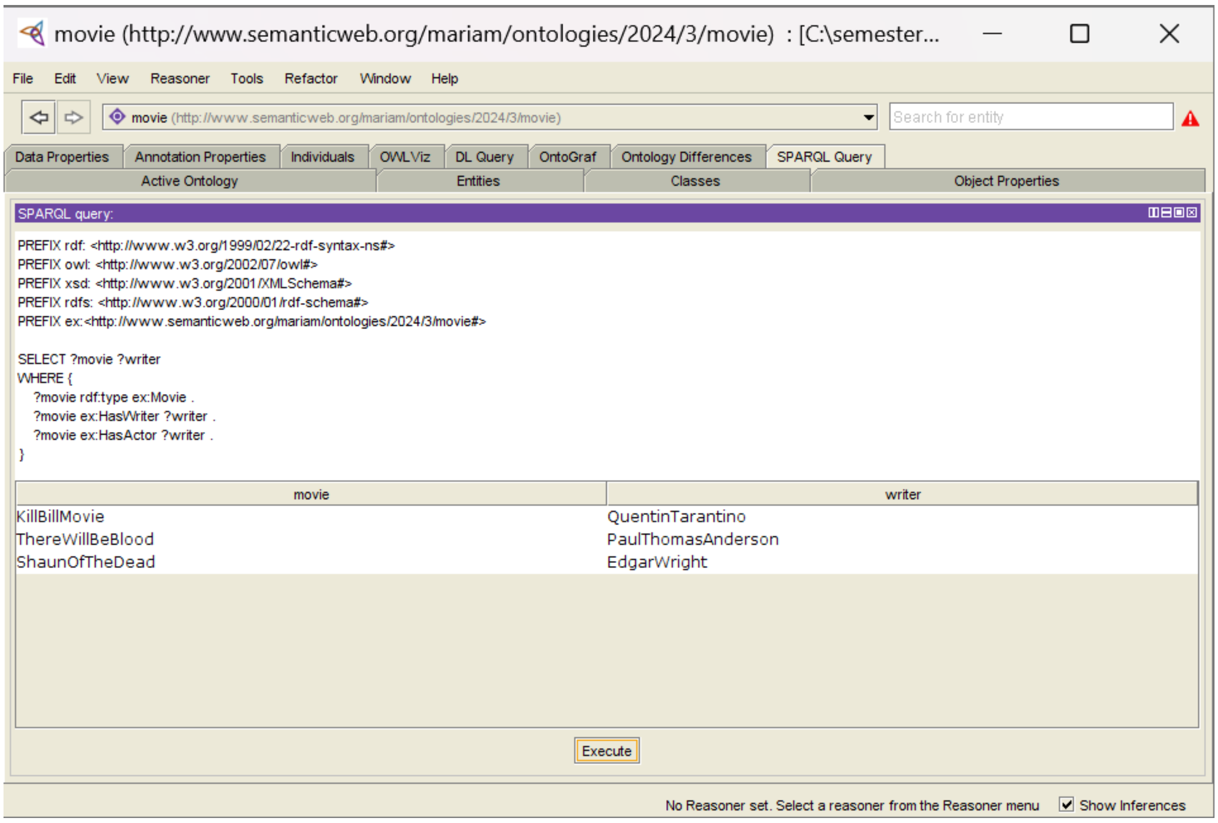

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\semester...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Active Ontology Entities Classes Object Properties
SPARQL query:
Execute
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?nationality
WHERE {
   ex:PulpFictionMovie rdf:type ex:Movie .
   ex:PulpFictionMovie ex:HasDirector ?director .
   ?director ex:nationality ?nationality .
}
```

nationality
"American"@

9. In this example, We List all movies where the writer also appeared as an actor and the writer/actor name.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>


SELECT ?movie ?writer

WHERE {

   ?movie rdf:type ex:Movie .

   ?movie ex:HasWriter ?writer .

   ?movie ex:HasActor ?writer .

}
```

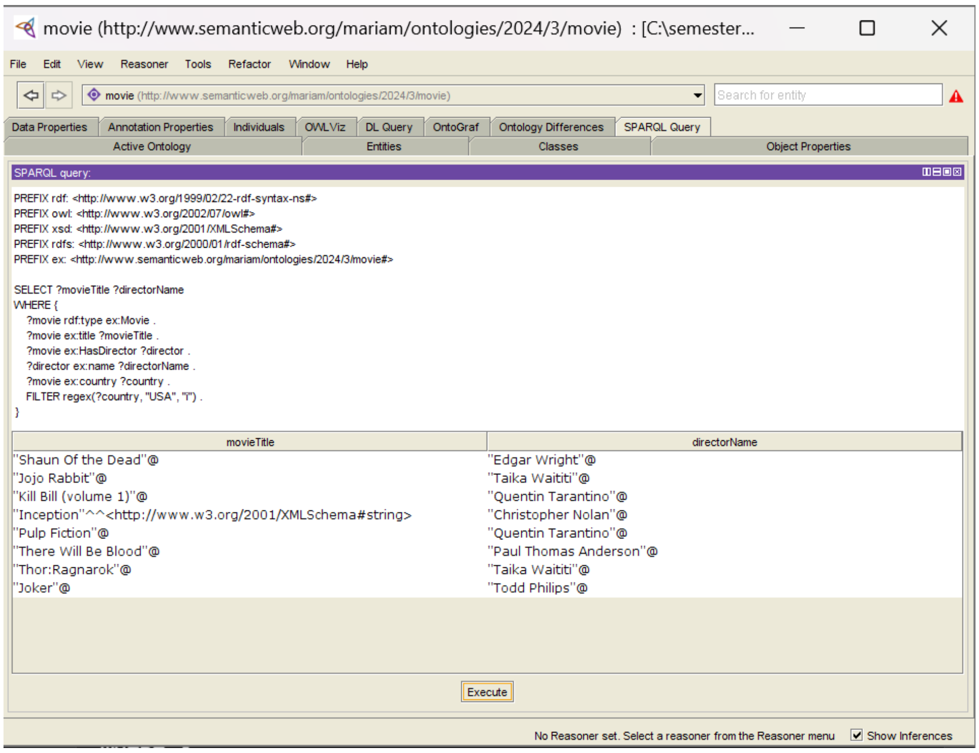

movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\semester...
File Edit View Reasoner Tools Refactor Window Help
movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)
Search for entity
Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Active Ontology Entities Classes Object Properties
SPARQL query:
Execute
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?movie ?writer
WHERE {
  ?movie rdf:type ex:Movie .
  ?movie ex:HasWriter ?writer .
  ?movie ex:HasActor ?writer .
}
```

movie	writer
KillBillMovie	QuentinTarantino
ThereWillBeBlood	PaulThomasAnderson
ShaunOfTheDead	EdgarWright

10. This query provides a way to retrieve information about movies released in a specific country along with their directors (e.g., USA).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ex: <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>


SELECT ?movieTitle ?directorName

WHERE {

   ?movie rdf:type ex:Movie .

   ?movie ex:title ?movieTitle .

   ?movie ex:HasDirector ?director .

   ?director ex:name ?directorName .

   ?movie ex:country ?country .

   FILTER regex(?country, "USA", "i") .

}
```

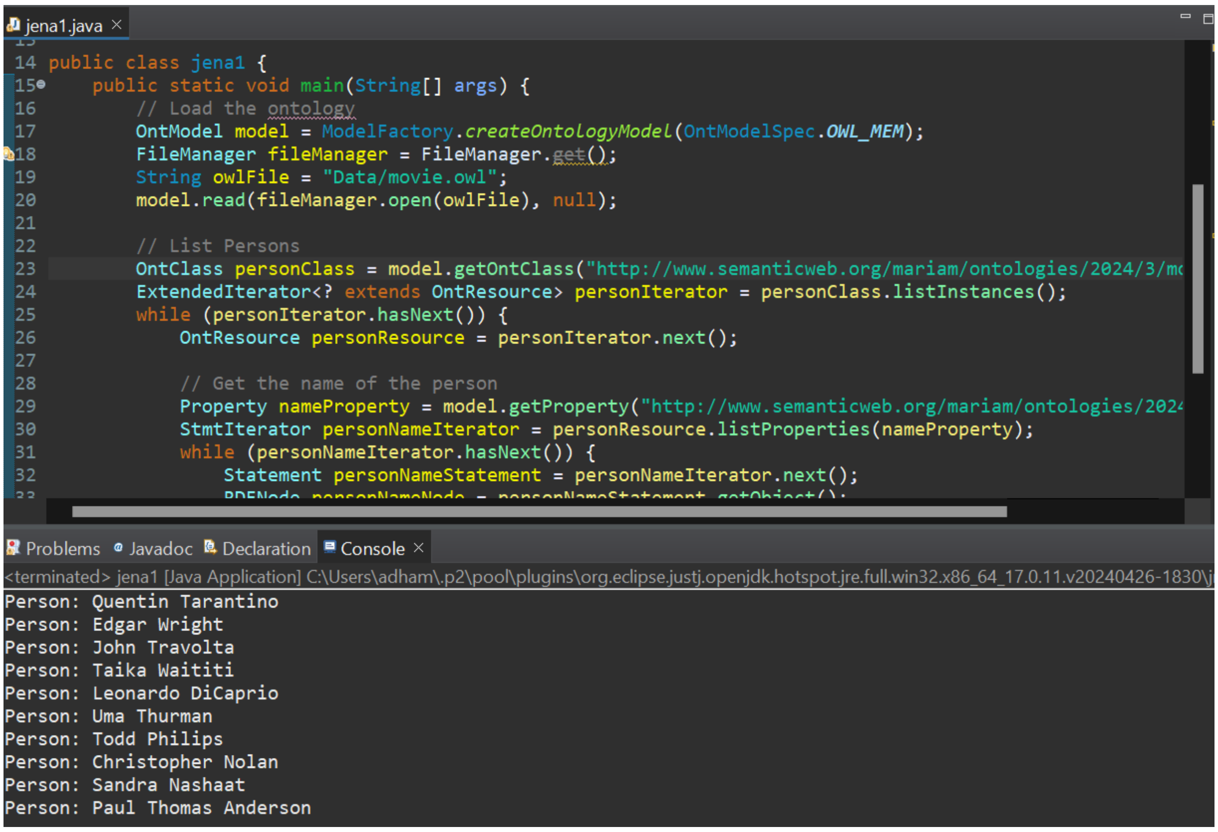


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?movieTitle ?directorName
WHERE {
   ?movie rdf:type ex:Movie .
   ?movie ex:title ?movieTitle .
   ?movie ex:HasDirector ?director .
   ?director ex:name ?directorName .
   ?movie ex:country ?country .
   FILTER regex(?country, "USA", "i") .
}
```

movieTitle	directorName
"Shaun Of the Dead"@	"Edgar Wright"@
"Jojo Rabbit"@	"Taika Waititi"@
"Kill Bill (volume 1)"@	"Quentin Tarantino"@
"Inception"^^<http://www.w3.org/2001/XMLSchema#string>	"Christopher Nolan"@
"Pulp Fiction"@	"Quentin Tarantino"@
"There Will Be Blood"@	"Paul Thomas Anderson"@
"Thor:Ragnarok"@	"Taika Waititi"@
"Joker"@	"Todd Philips"@

Part IV: Manipulating the ontology using Jena

1. **Create a java program (Jena1.java) that loads the ontology and displays all the Persons (without using queries, without inference).**



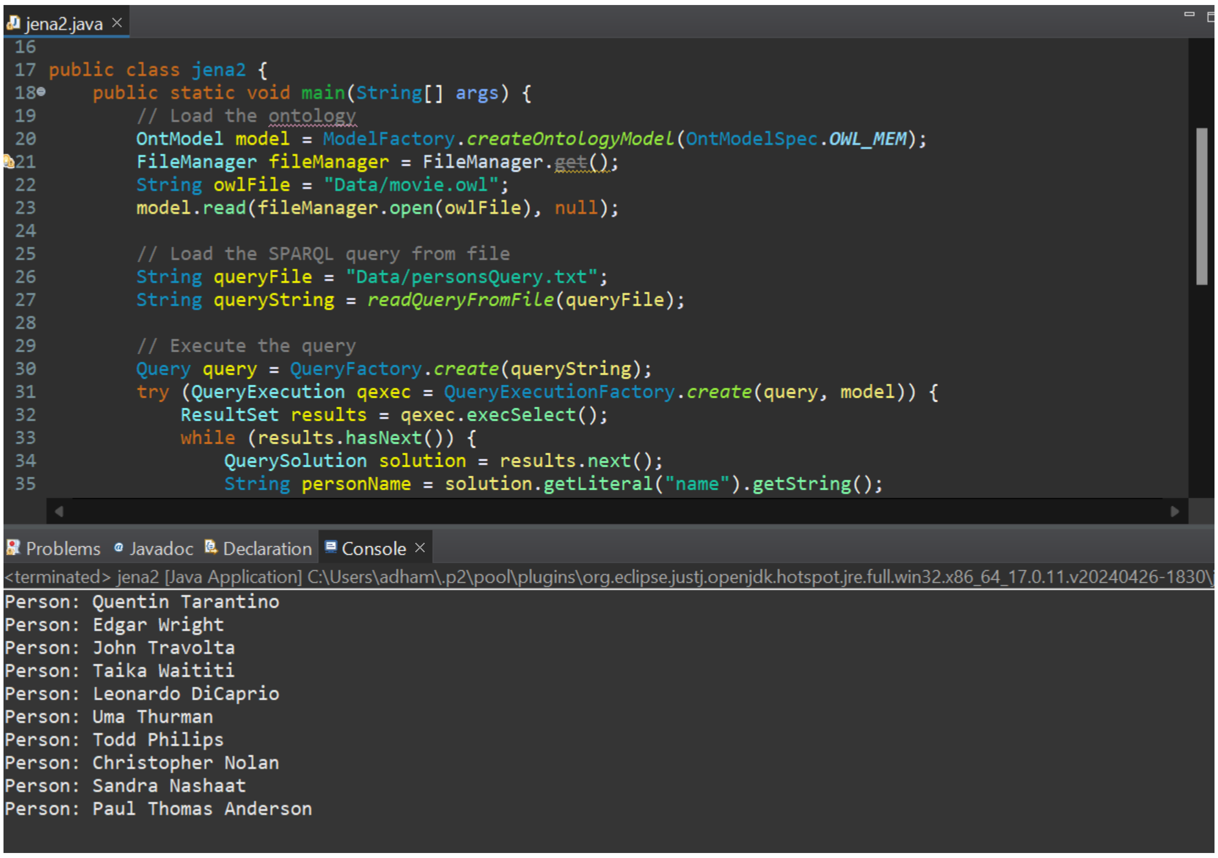
jena1.java

```java
public class jena1 {
    public static void main(String[] args) {
        // Load the ontology
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        FileManager fileManager = FileManager.get();
        String owlFile = "Data/movie.owl";
        model.read(fileManager.open(owlFile), null);

        // List Persons
        OntClass personClass = model.getOntClass("http://www.semanticweb.org/mariam/ontologies/2024/3/mo
        ExtendedIterator<? extends OntResource> personIterator = personClass.listInstances();
        while (personIterator.hasNext()) {
            OntResource personResource = personIterator.next();

            // Get the name of the person
            Property nameProperty = model.getProperty("http://www.semanticweb.org/mariam/ontologies/2024
            StmtIterator personNameIterator = personResource.listProperties(nameProperty);
            while (personNameIterator.hasNext()) {
                Statement personNameStatement = personNameIterator.next();
                RDFNode personNameNode = personNameStatement.getObject();
```

```
<terminated> jena1 [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\j
Person: Quentin Tarantino
Person: Edgar Wright
Person: John Travolta
Person: Taika Waititi
Person: Leonardo DiCaprio
Person: Uma Thurman
Person: Todd Philips
Person: Christopher Nolan
Person: Sandra Nashaat
Person: Paul Thomas Anderson
```

jena 1

2. **Create a java program (Jena2.java) that loads the ontology and displays all the Persons (using a query, without inference). Create the used query in text file under the data folder.**



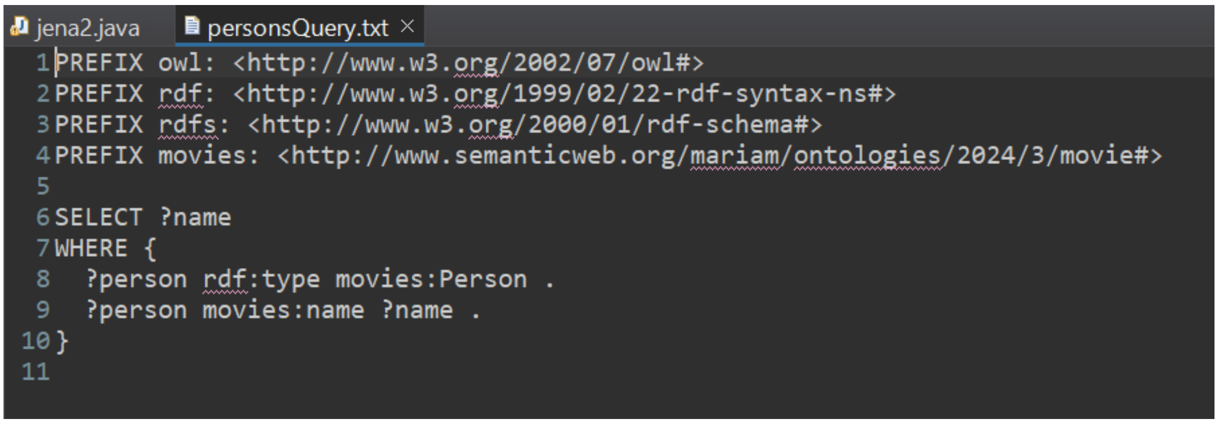
jena2.java ×

```java
public class jena2 {
    public static void main(String[] args) {
        // Load the ontology
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        FileManager fileManager = FileManager.get();
        String owlFile = "Data/movie.owl";
        model.read(fileManager.open(owlFile), null);

        // Load the SPARQL query from file
        String queryFile = "Data/personsQuery.txt";
        String queryString = readQueryFromFile(queryFile);

        // Execute the query
        Query query = QueryFactory.create(queryString);
        try (QueryExecution qexec = QueryExecutionFactory.create(query, model)) {
            ResultSet results = qexec.execSelect();
            while (results.hasNext()) {
                QuerySolution solution = results.next();
                String personName = solution.getLiteral("name").getString();
```

```
<terminated> jena2 [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\
Person: Quentin Tarantino
Person: Edgar Wright
Person: John Travolta
Person: Taika Waititi
Person: Leonardo DiCaprio
Person: Uma Thurman
Person: Todd Philips
Person: Christopher Nolan
Person: Sandra Nashaat
Person: Paul Thomas Anderson
```

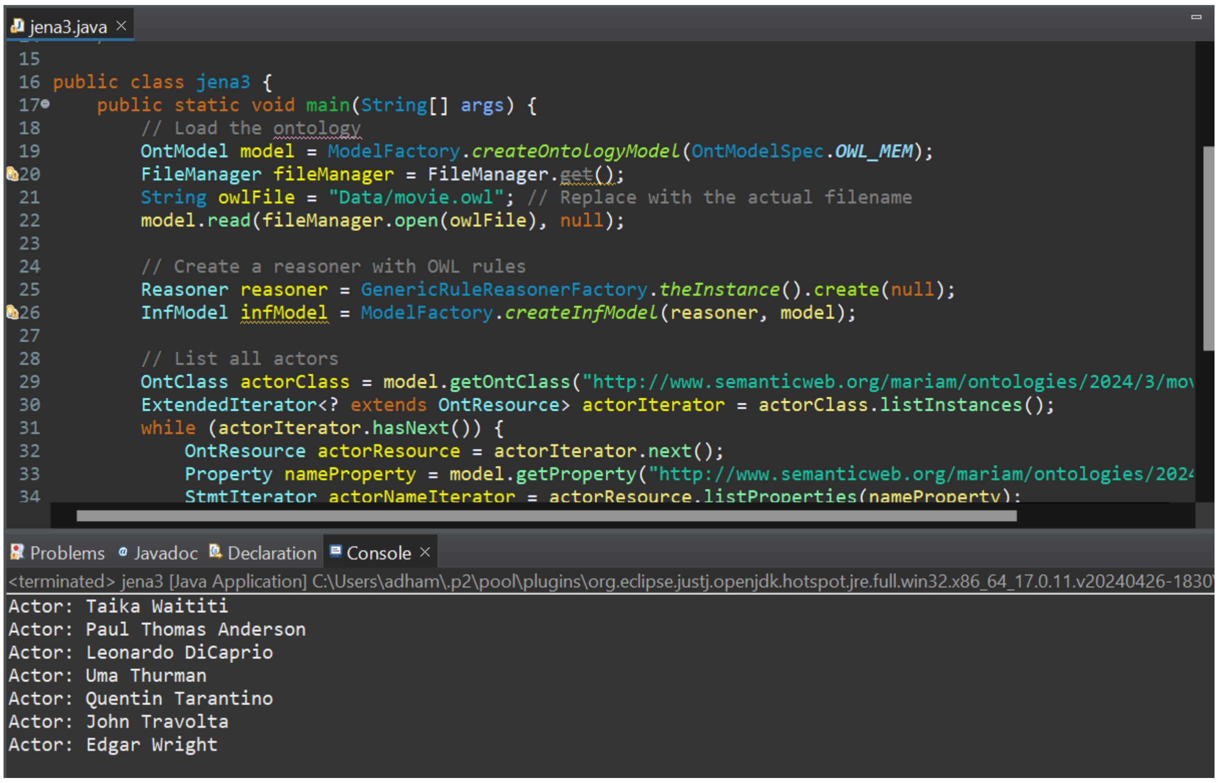
jena 2

Query for jena2:



```sparql
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX movies: <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>

SELECT ?name
WHERE {
  ?person rdf:type movies:Person .
  ?person movies:name ?name .
}
```

3. **Create a java program (Jena3.java) that loads the ontology and displays all the Actors (without using queries, using inference). To load the inferred model, use the JenaEngine.readInferencedModelFromRuleFile method and use owl rules**



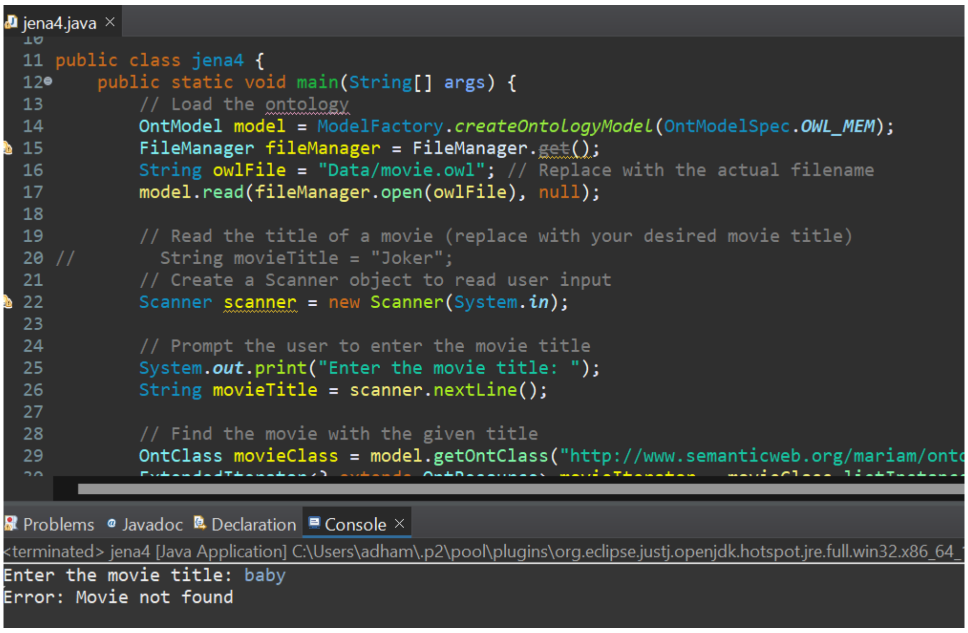
```java

public class jena3 {
    public static void main(String[] args) {
        // Load the ontology
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        FileManager fileManager = FileManager.get();
        String owlFile = "Data/movie.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);

        // Create a reasoner with OWL rules
        Reasoner reasoner = GenericRuleReasonerFactory.theInstance().create(null);
        InfModel infModel = ModelFactory.createInfModel(reasoner, model);

        // List all actors
        OntClass actorClass = model.getOntClass("http://www.semanticweb.org/mariam/ontologies/2024/3/mov
        ExtendedIterator<? extends OntResource> actorIterator = actorClass.listInstances();
        while (actorIterator.hasNext()) {
            OntResource actorResource = actorIterator.next();
            Property nameProperty = model.getProperty("http://www.semanticweb.org/mariam/ontologies/2024
            StmtIterator actorNameIterator = actorResource.listProperties(nameProperty);
```

```
<terminated> jena3 [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830
Actor: Taika Waititi
Actor: Paul Thomas Anderson
Actor: Leonardo DiCaprio
Actor: Uma Thurman
Actor: Quentin Tarantino
Actor: John Travolta
Actor: Edgar Wright
```

jena 3

4. Create a java program (Jena4.java) that:
    a. Reads a name of a movie
    b. If it doesn't exist displays an error message
    c. Else, display its year, country, genres and actors

Movie not found:

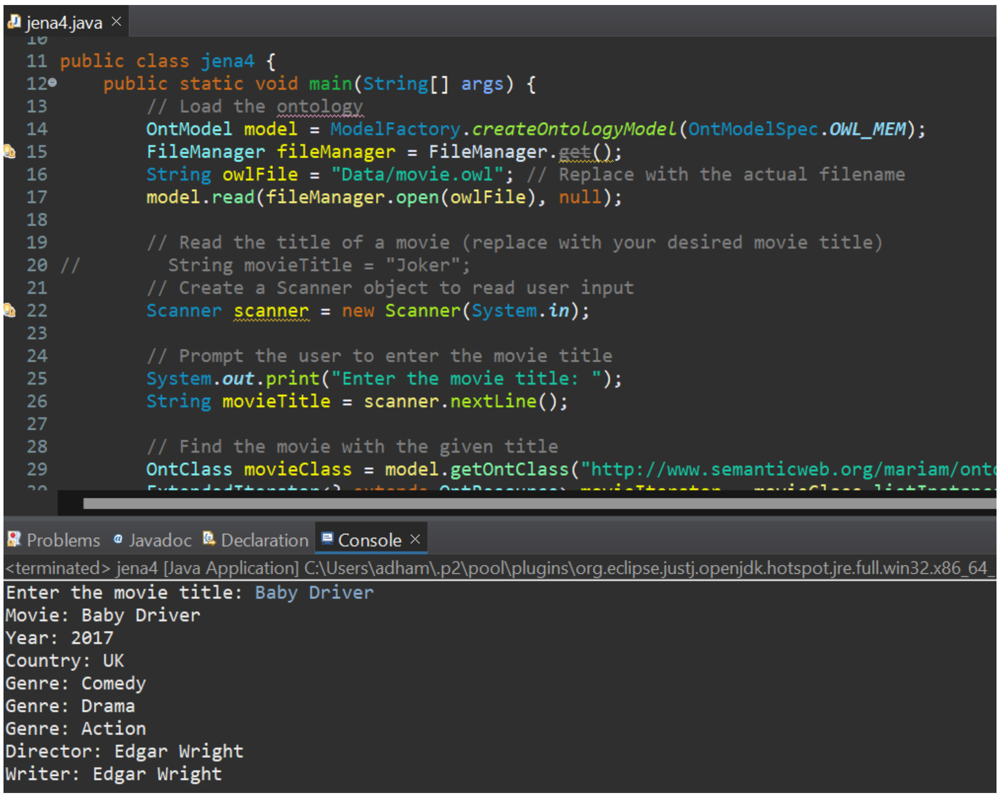


```java
public class jena4 {
    public static void main(String[] args) {
        // Load the ontology
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        FileManager fileManager = FileManager.get();
        String owlFile = "Data/movie.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);

        // Read the title of a movie (replace with your desired movie title)
//        String movieTitle = "Joker";
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the movie title
        System.out.print("Enter the movie title: ");
        String movieTitle = scanner.nextLine();

        // Find the movie with the given title
        OntClass movieClass = model.getOntClass("http://www.semanticweb.org/mariam/onto
```

```
<terminated> jena4 [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_
Enter the movie title: baby
Error: Movie not found
```

jena 4

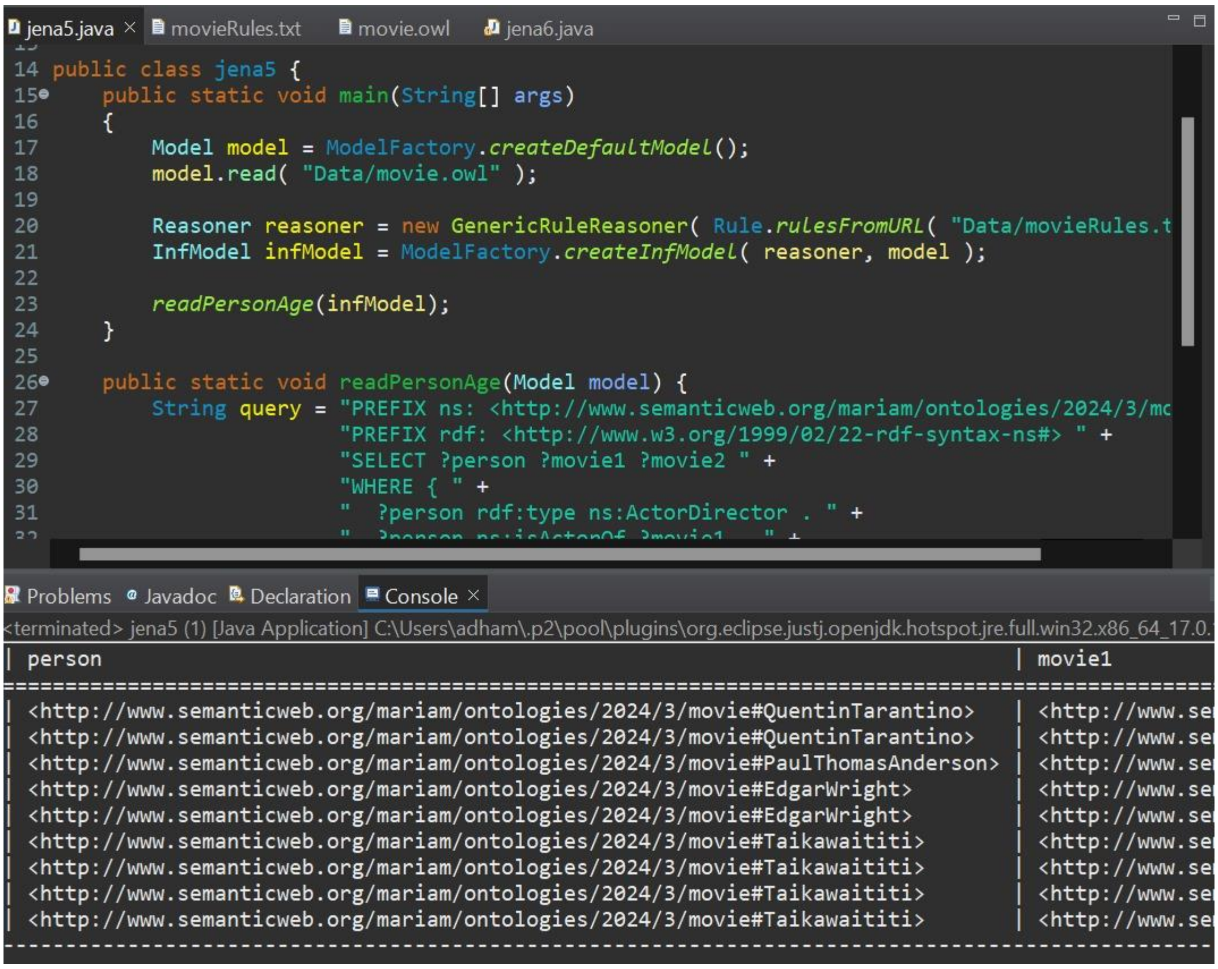
Movie found:



```java
public class jena4 {
    public static void main(String[] args) {
        // Load the ontology
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        FileManager fileManager = FileManager.get();
        String owlFile = "Data/movie.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);

        // Read the title of a movie (replace with your desired movie title)
//        String movieTitle = "Joker";
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the movie title
        System.out.print("Enter the movie title: ");
        String movieTitle = scanner.nextLine();

        // Find the movie with the given title
        OntClass movieClass = model.getOntClass("http://www.semanticweb.org/mariam/onto
```

```
<terminated> jena4 [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_
Enter the movie title: Baby Driver
Movie: Baby Driver
Year: 2017
Country: UK
Genre: Comedy
Genre: Drama
Genre: Action
Director: Edgar Wright
Writer: Edgar Wright
```

5. **Create a java program (Jena5.java) that displays all persons that are actors and directors. Do this using a rule that defines a new class ActorDirector. The rule file must be saved in the data folder.**

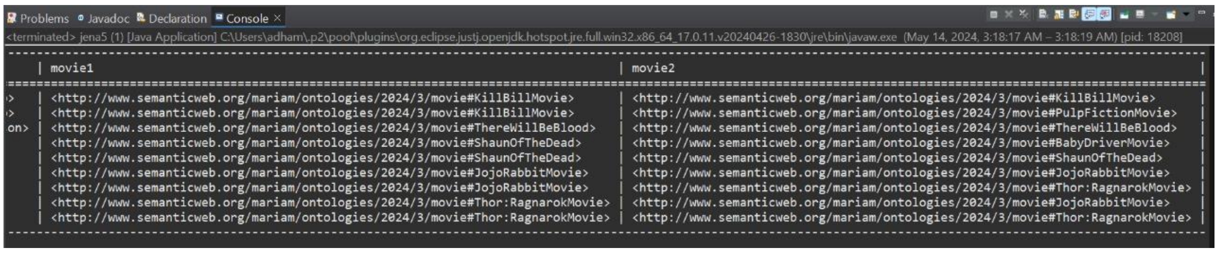


```java
public class jena5 {
    public static void main(String[] args)
    {
        Model model = ModelFactory.createDefaultModel();
        model.read( "Data/movie.owl" );

        Reasoner reasoner = new GenericRuleReasoner( Rule.rulesFromURL( "Data/movieRules.t
        InfModel infModel = ModelFactory.createInfModel( reasoner, model );

        readPersonAge(infModel);
    }

    public static void readPersonAge(Model model) {
        String query = "PREFIX ns: <http://www.semanticweb.org/mariam/ontologies/2024/3/mc
                       "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
                       "SELECT ?person ?movie1 ?movie2 " +
                       "WHERE { " +
                       "  ?person rdf:type ns:ActorDirector . " +
                       "  ?person ns:isActorOf ?movie1 .  " +
```

```
<terminated> jena5 (1) [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.
| person                                                                 | movie1
==========================================================================================
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#QuentinTarantino>    | <http://www.se
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#QuentinTarantino>    | <http://www.se
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#PaulThomasAnderson> | <http://www.se
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#EdgarWright>         | <http://www.se
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#EdgarWright>         | <http://www.se
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Taikawaititi>        | <http://www.se
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Taikawaititi>        | <http://www.se
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Taikawaititi>        | <http://www.se
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Taikawaititi>        | <http://www.se
------------------------------------------------------------------------------------------
```

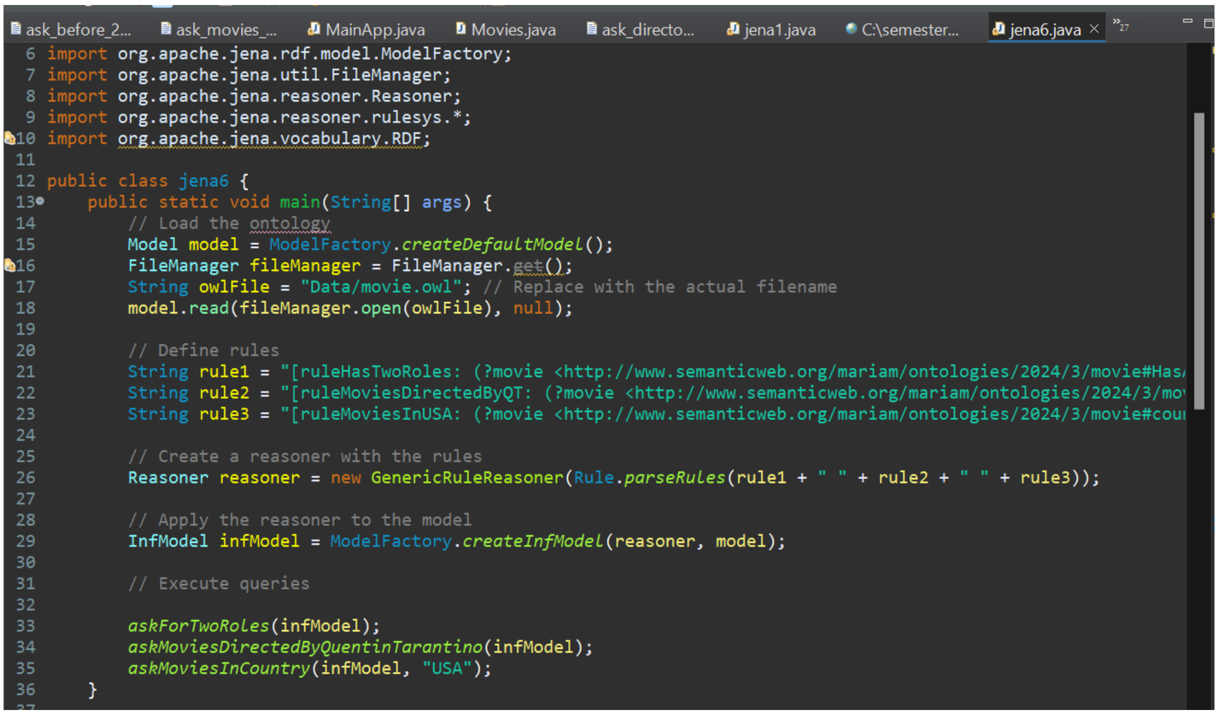
jena 5

jena5 Rule:



```
<terminated> jena5 (1) [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe  (May 14, 2024, 3:18:17 AM – 3:18:19 AM) [pid: 18208]
-------------------------------------------------------------------------------------------------------------------------------------------------------------
    | movie1                                                                   | movie2                                                                   |
=============================================================================================================================================================
>   | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#KillBillMovie>       | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#KillBillMovie>        |
>   | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#KillBillMovie>       | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#PulpFictionMovie>     |
on> | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#ThereWillBeBlood>    | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#ThereWillBeBlood>     |
    | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#ShaunOfTheDead>      | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#BabyDriverMovie>      |
    | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#ShaunOfTheDead>      | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#ShaunOfTheDead>       |
    | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#JojoRabbitMovie>     | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#JojoRabbitMovie>      |
    | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#JojoRabbitMovie>     | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Thor:RagnarokMovie>   |
    | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Thor:RagnarokMovie>  | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#JojoRabbitMovie>      |
    | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Thor:RagnarokMovie>  | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Thor:RagnarokMovie>   |
-------------------------------------------------------------------------------------------------------------------------------------------------------------
```

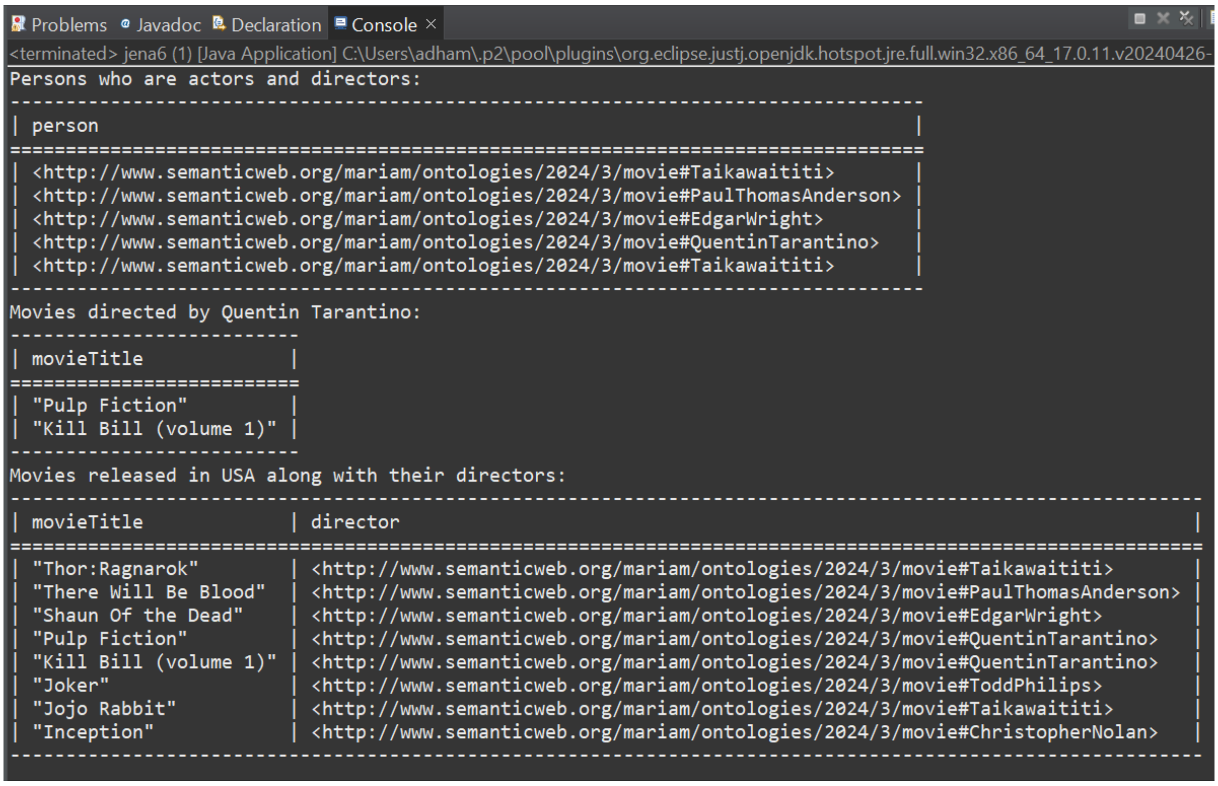
6. Specify 3 different rules and implement them in a java program (Jena6.java)



```java
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.util.FileManager;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.rulesys.*;
import org.apache.jena.vocabulary.RDF;

public class jena6 {
    public static void main(String[] args) {
        // Load the ontology
        Model model = ModelFactory.createDefaultModel();
        FileManager fileManager = FileManager.get();
        String owlFile = "Data/movie.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);

        // Define rules
        String rule1 = "[ruleHasTwoRoles: (?movie <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Has
        String rule2 = "[ruleMoviesDirectedByQT: (?movie <http://www.semanticweb.org/mariam/ontologies/2024/3/mov
        String rule3 = "[ruleMoviesInUSA: (?movie <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#cou

        // Create a reasoner with the rules
        Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rule1 + " " + rule2 + " " + rule3));

        // Apply the reasoner to the model
        InfModel infModel = ModelFactory.createInfModel(reasoner, model);

        // Execute queries

        askForTwoRoles(infModel);
        askMoviesDirectedByQuentinTarantino(infModel);
        askMoviesInCountry(infModel, "USA");
    }
```

jena 6

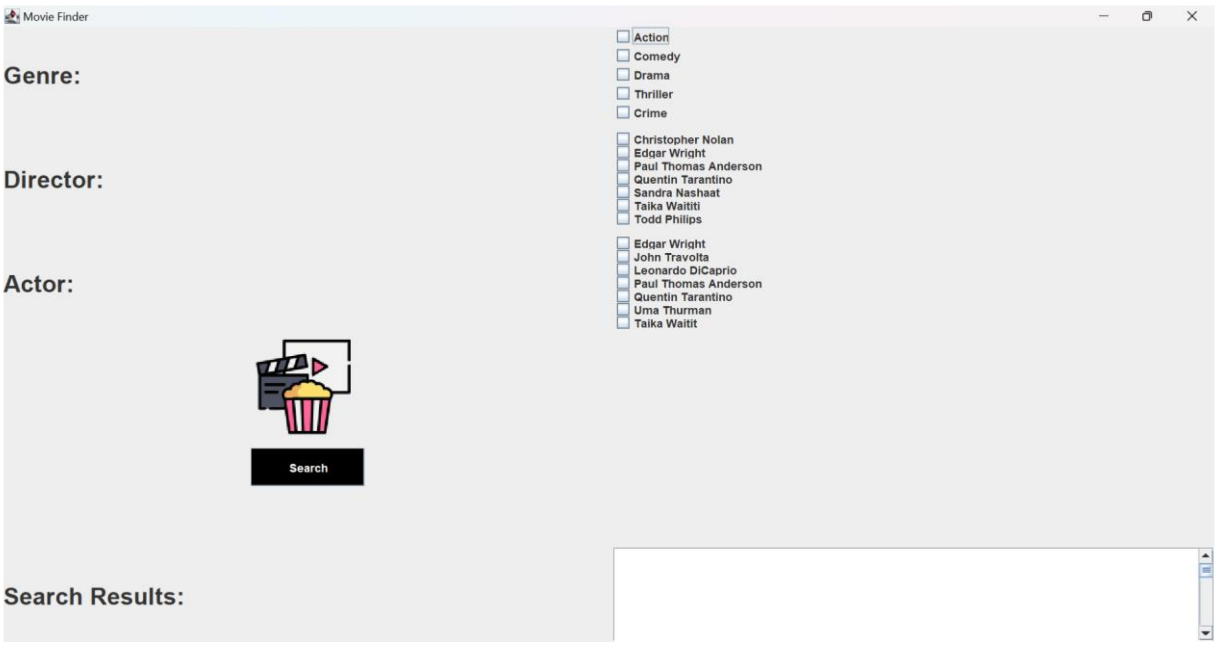
jena6 full Output:



```
<terminated> jena6 (1) [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-
Persons who are actors and directors:
---------------------------------------------------------------------------------
| person                                                                        |
=================================================================================
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Taikawaititi>      |
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#PaulThomasAnderson> |
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#EdgarWright>       |
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#QuentinTarantino>  |
| <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Taikawaititi>      |
---------------------------------------------------------------------------------
Movies directed by Quentin Tarantino:
-------------------------
| movieTitle            |
=========================
| "Pulp Fiction"        |
| "Kill Bill (volume 1)" |
-------------------------
Movies released in USA along with their directors:
--------------------------------------------------------------------------------------------------------------
| movieTitle             | director                                                                       |
==============================================================================================================
| "Thor:Ragnarok"        | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Taikawaititi>       |
| "There Will Be Blood"  | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#PaulThomasAnderson> |
| "Shaun Of the Dead"    | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#EdgarWright>        |
| "Pulp Fiction"         | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#QuentinTarantino>   |
| "Kill Bill (volume 1)" | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#QuentinTarantino>   |
| "Joker"                | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#ToddPhilips>        |
| "Jojo Rabbit"          | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Taikawaititi>       |
| "Inception"            | <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#ChristopherNolan>   |
--------------------------------------------------------------------------------------------------------------
```

Part V: Java application

1. GUI



2. Application Overview

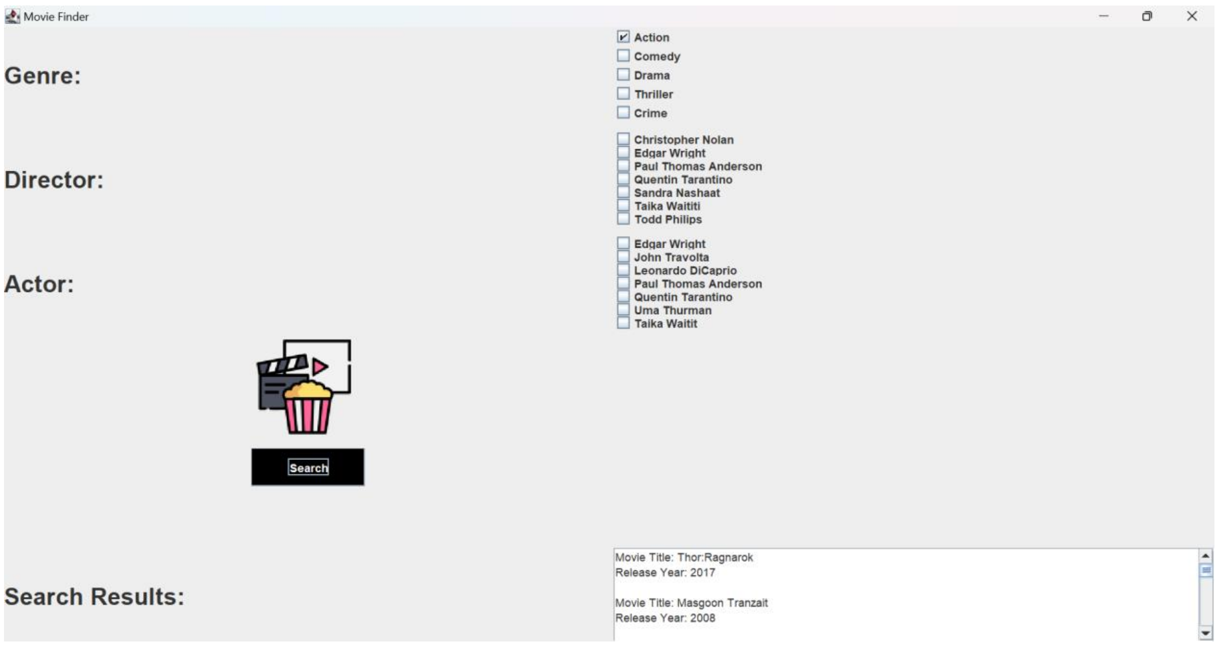
This java application (Movie Finder) returns you a list of films based on the included/excluded actors, directors and genres.

The boxes you check are what you want to include and unchecked boxes are excluded.

You can Check up to 2 boxes in each category (we assumed this logic because dataset is limited so cheking more than 2 will probably output all availble movies), then press search and the movies that fullfill your selections will appear in seach resulst scroll box on the botttom right.

3. Application Quick Manual Test Cases

Here are some samples of test cases we did to make sure the application is working as expected:

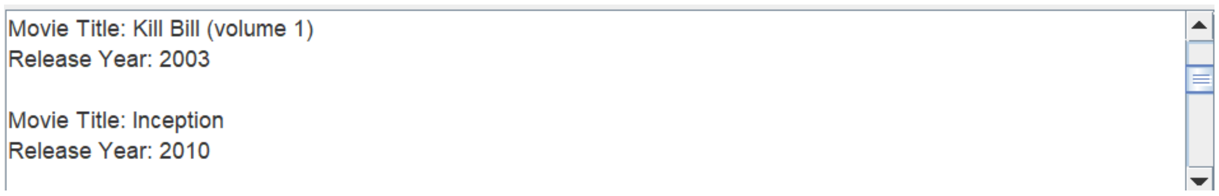


☑ Action
☐ Comedy
☐ Drama
☐ Thriller
☐ Crime

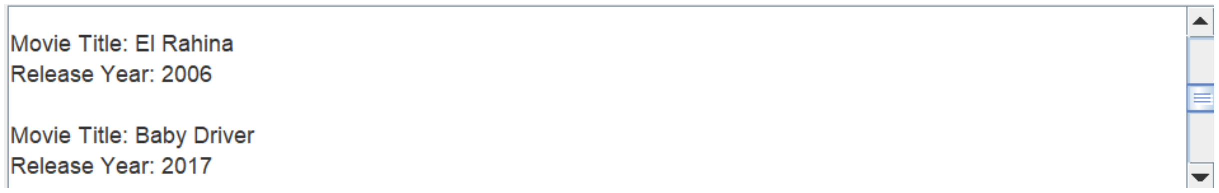
☐ Christopher Nolan
☐ Edgar Wright
☐ Paul Thomas Anderson
☐ Quentin Tarantino
☐ Sandra Nashaat
☐ Taika Waititi
☐ Todd Philips

☐ Edgar Wright
☐ John Travolta
☐ Leonardo DiCaprio
☐ Paul Thomas Anderson
☐ Quentin Tarantino
☐ Uma Thurman
☐ Taika Waitit

```
Movie Title: Thor:Ragnarok
Release Year: 2017

Movie Title: Masgoon Tranzait
Release Year: 2008
```

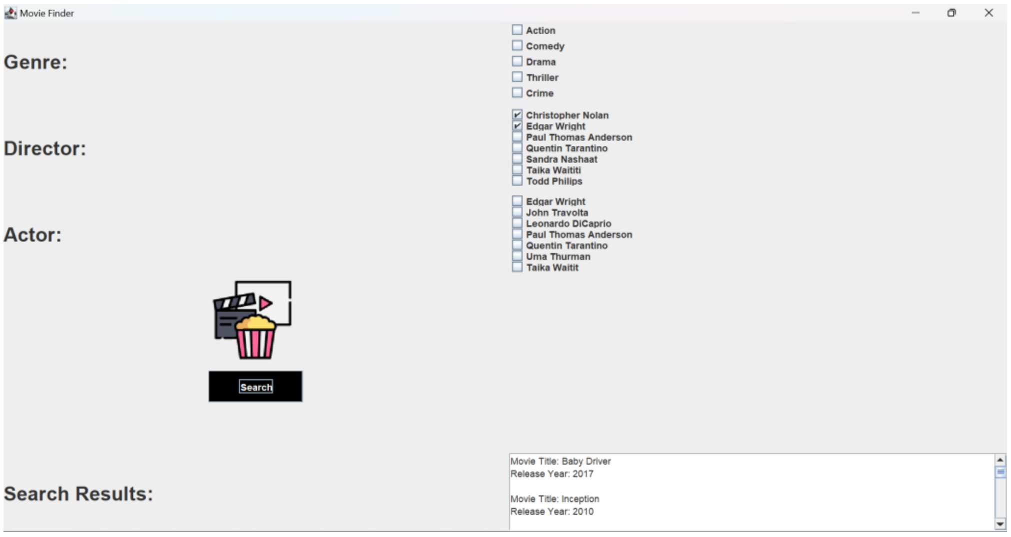


```
Movie Title: Kill Bill (volume 1)
Release Year: 2003

Movie Title: Inception
Release Year: 2010
```



```
Movie Title: El Rahina
Release Year: 2006

Movie Title: Baby Driver
Release Year: 2017
```

In the previous example, we selected movies of the action genre regardless of the actors and directors. The output displayed six movies that indeed belong to the action genre. Any movie that wasn't of the action genre didn't appear.


Movie Finder

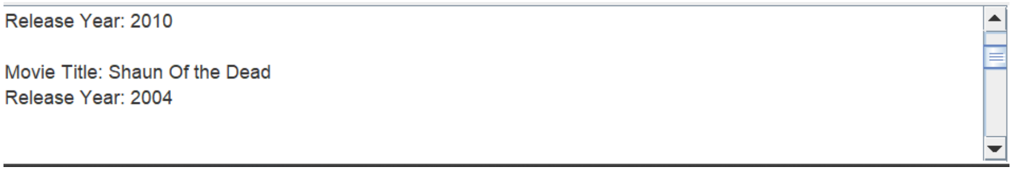
Genre:
- ☐ Action
- ☐ Comedy
- ☐ Drama
- ☐ Thriller
- ☐ Crime

Director:
- ☑ Christopher Nolan
- ☑ Edgar Wright
- ☐ Paul Thomas Anderson
- ☐ Quentin Tarantino
- ☐ Sandra Nashaat
- ☐ Taika Waititi
- ☐ Todd Philips

Actor:
- ☐ Edgar Wright
- ☐ John Travolta
- ☐ Leonardo DiCaprio
- ☐ Paul Thomas Anderson
- ☐ Quentin Tarantino
- ☐ Uma Thurman
- ☐ Taika Waitit

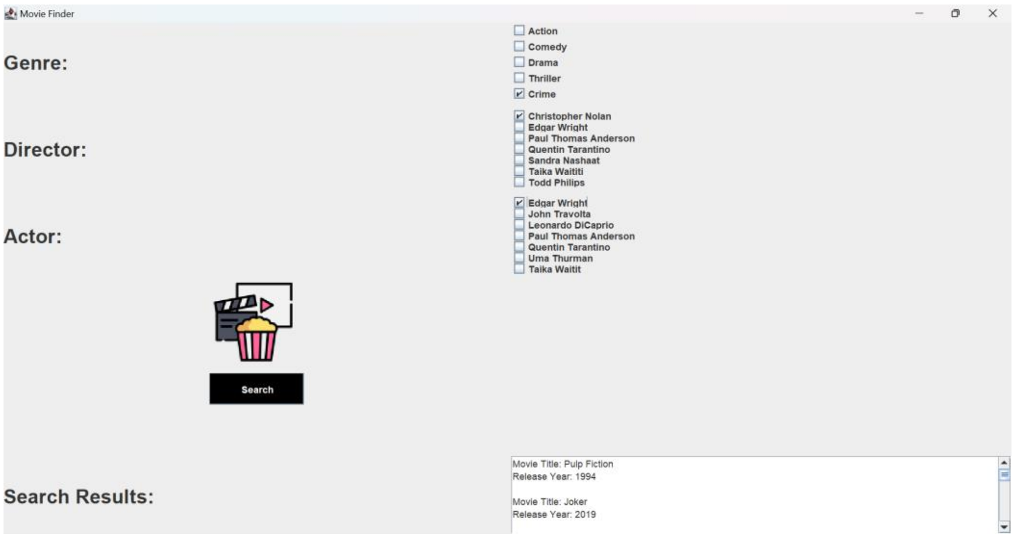
Search Results:

```
Movie Title: Baby Driver
Release Year: 2017

Movie Title: Inception
Release Year: 2010
```



```
Release Year: 2010

Movie Title: Shaun Of the Dead
Release Year: 2004
```

Here is another test case where we only included movies directed by Christopher Nolan and Edgar Wright.


Movie Finder

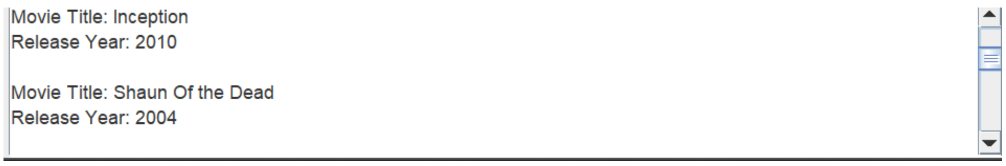
Genre:
- ☐ Action
- ☐ Comedy
- ☐ Drama
- ☐ Thriller
- ☑ Crime

Director:
- ☑ Christopher Nolan
- ☐ Edgar Wright
- ☐ Paul Thomas Anderson
- ☐ Quentin Tarantino
- ☐ Sandra Nashaat
- ☐ Taika Waititi
- ☐ Todd Philips

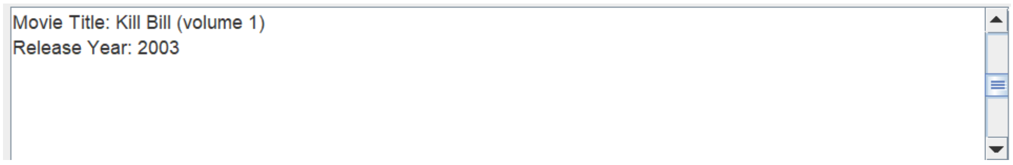
Actor:
- ☑ Edgar Wright
- ☐ John Travolta
- ☐ Leonardo DiCaprio
- ☐ Paul Thomas Anderson
- ☐ Quentin Tarantino
- ☐ Uma Thurman
- ☐ Taika Waitit

Search Results:

```
Movie Title: Pulp Fiction
Release Year: 1994

Movie Title: Joker
Release Year: 2019
```

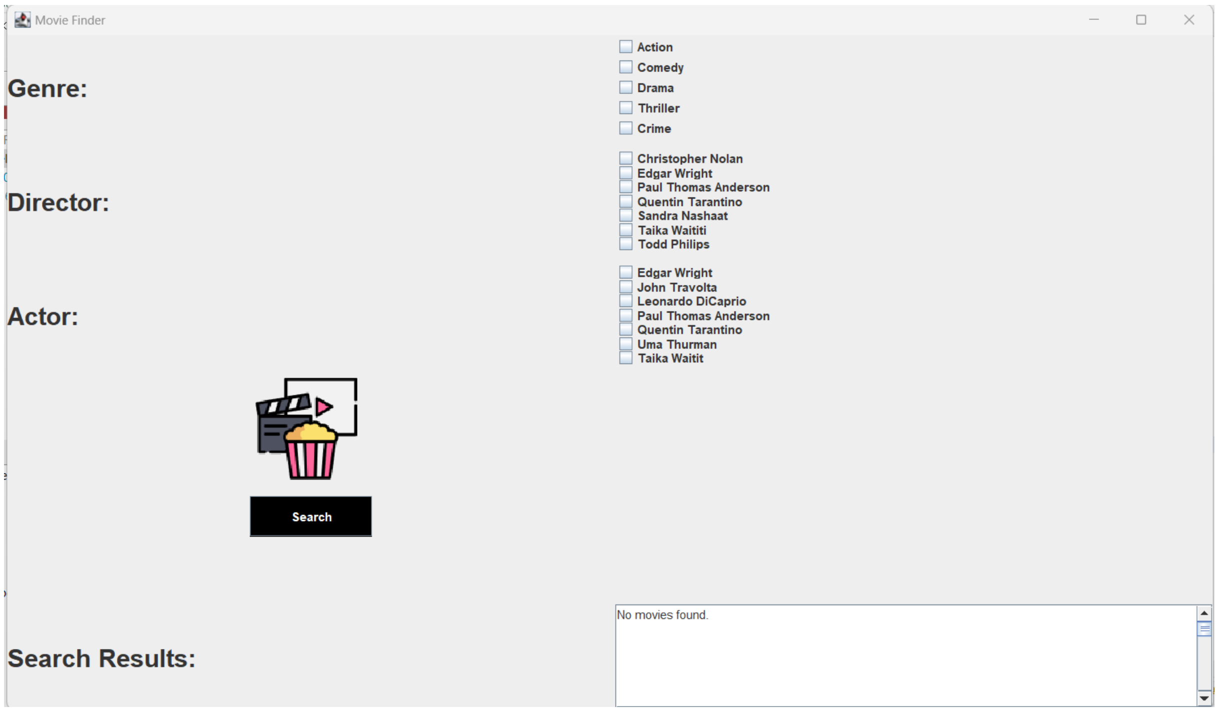


```
Movie Title: Inception
Release Year: 2010

Movie Title: Shaun Of the Dead
Release Year: 2004
```



```
Movie Title: Kill Bill (volume 1)
Release Year: 2003
```

As shown here, we included one option from each category to ensure they work well together, and the output is as expected.



Movie Finder

Genre:
- [ ] Action
- [ ] Comedy
- [ ] Drama
- [ ] Thriller
- [ ] Crime

Director:
- [ ] Christopher Nolan
- [ ] Edgar Wright
- [ ] Paul Thomas Anderson
- [ ] Quentin Tarantino
- [ ] Sandra Nashaat
- [ ] Taika Waititi
- [ ] Todd Philips

Actor:
- [ ] Edgar Wright
- [ ] John Travolta
- [ ] Leonardo DiCaprio
- [ ] Paul Thomas Anderson
- [ ] Quentin Tarantino
- [ ] Uma Thurman
- [ ] Taika Waitit

Search

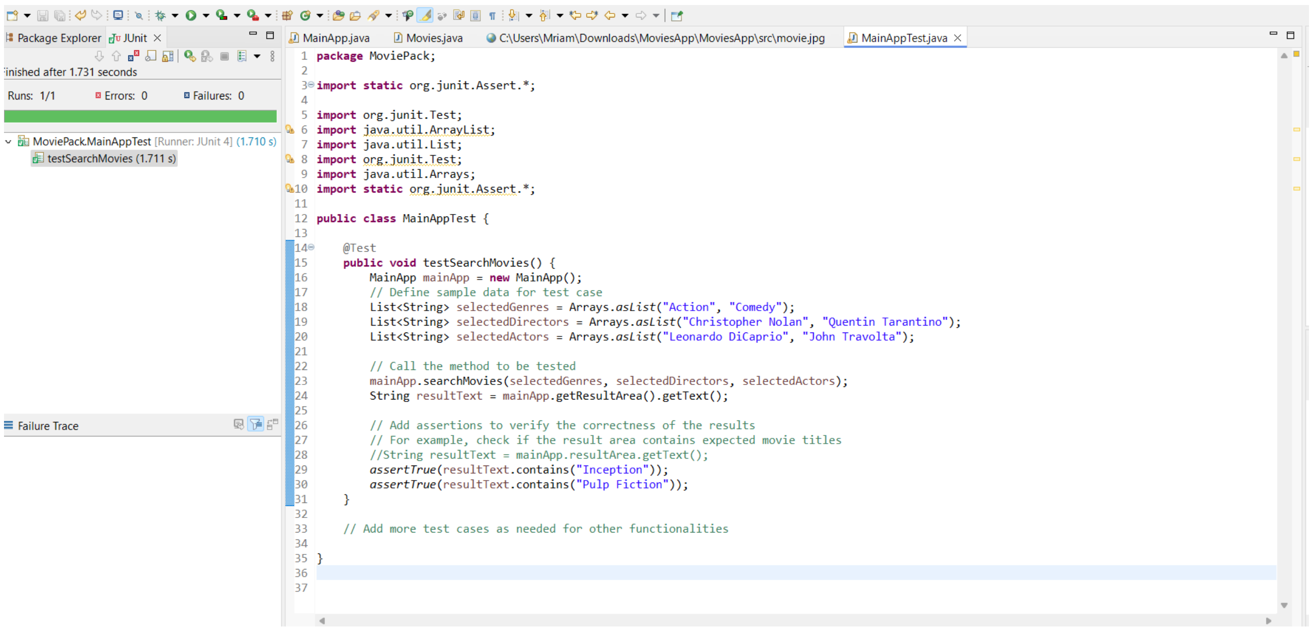
Search Results:

```
No movies found.
```

If the user makes no selection it appears as "No movies found" as output.

Testing : Ontology Consistency

Test case 1



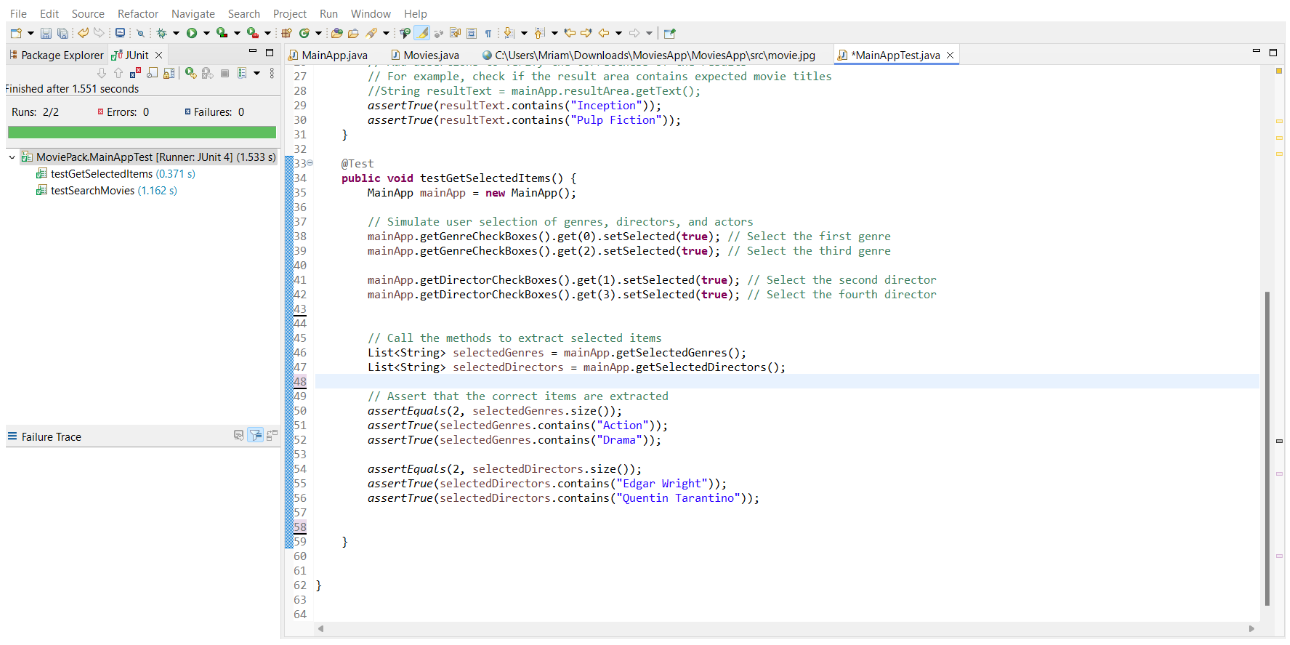
```
Finished after 1.731 seconds
Runs: 1/1    Errors: 0    Failures: 0
MoviePack.MainAppTest [Runner: JUnit 4] (1.710 s)
    testSearchMovies (1.711 s)
```

```java
package MoviePack;

import static org.junit.Assert.*;

import org.junit.Test;
import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import java.util.Arrays;
import static org.junit.Assert.*;

public class MainAppTest {

    @Test
    public void testSearchMovies() {
        MainApp mainApp = new MainApp();
        // Define sample data for test case
        List<String> selectedGenres = Arrays.asList("Action", "Comedy");
        List<String> selectedDirectors = Arrays.asList("Christopher Nolan", "Quentin Tarantino");
        List<String> selectedActors = Arrays.asList("Leonardo DiCaprio", "John Travolta");

        // Call the method to be tested
        mainApp.searchMovies(selectedGenres, selectedDirectors, selectedActors);
        String resultText = mainApp.getResultArea().getText();

        // Add assertions to verify the correctness of the results
        // For example, check if the result area contains expected movie titles
        //String resultText = mainApp.resultArea.getText();
        assertTrue(resultText.contains("Inception"));
        assertTrue(resultText.contains("Pulp Fiction"));
    }

    // Add more test cases as needed for other functionalities

}
```

This test case is designed to verify the functionality of the **searchMovies** method in the **MainApp** class. The **searchMovies** method is responsible for querying movies based on selected genres, directors, and actors, and displaying the results in the result area.

The **assertTrue** method is used to assert that the **resultText** contains the expected movie titles. If any of these assertions fail, it indicates that there is an issue with the **searchMovies** method or the way it processes the user input.

Test case 2



File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
MainApp.java
Movies.java
C:\Users\Mriam\Downloads\MoviesApp\MoviesApp\src\movie.jpg
*MainAppTest.java
Failure Trace

```
Finished after 1.551 seconds
Runs: 2/2    Errors: 0    Failures: 0

MoviePack.MainAppTest [Runner: JUnit 4] (1.533 s)
    testGetSelectedItems (0.371 s)
    testSearchMovies (1.162 s)
```

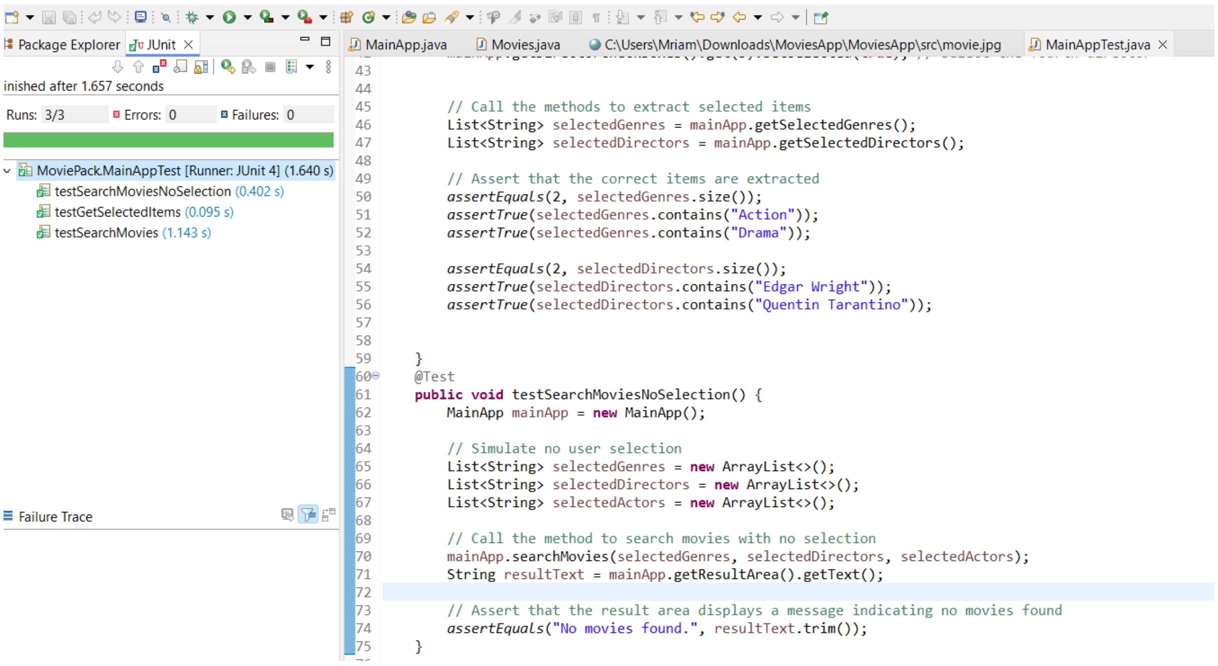
```java
        // For example, check if the result area contains expected movie titles
        //String resultText = mainApp.resultArea.getText();
        assertTrue(resultText.contains("Inception"));
        assertTrue(resultText.contains("Pulp Fiction"));
    }

    @Test
    public void testGetSelectedItems() {
        MainApp mainApp = new MainApp();

        // Simulate user selection of genres, directors, and actors
        mainApp.getGenreCheckBoxes().get(0).setSelected(true); // Select the first genre
        mainApp.getGenreCheckBoxes().get(2).setSelected(true); // Select the third genre

        mainApp.getDirectorCheckBoxes().get(1).setSelected(true); // Select the second director
        mainApp.getDirectorCheckBoxes().get(3).setSelected(true); // Select the fourth director


        // Call the methods to extract selected items
        List<String> selectedGenres = mainApp.getSelectedGenres();
        List<String> selectedDirectors = mainApp.getSelectedDirectors();

        // Assert that the correct items are extracted
        assertEquals(2, selectedGenres.size());
        assertTrue(selectedGenres.contains("Action"));
        assertTrue(selectedGenres.contains("Drama"));

        assertEquals(2, selectedDirectors.size());
        assertTrue(selectedDirectors.contains("Edgar Wright"));
        assertTrue(selectedDirectors.contains("Quentin Tarantino"));


    }


}
```

This test case, **testGetSelectedItems**, verifies that the **getSelectedGenres** and **getSelectedDirectors** methods in the **MainApp** class correctly extract the selected genres and directors from the user interface.

Assertion: The test asserts that the correct items are extracted. It checks:

- If the number of selected genres and directors matches the expected count.
- If the selected genres and directors contain the expected values ("Action", "Drama", "Edgar Wright", "Quentin Tarantino").

Test case 3



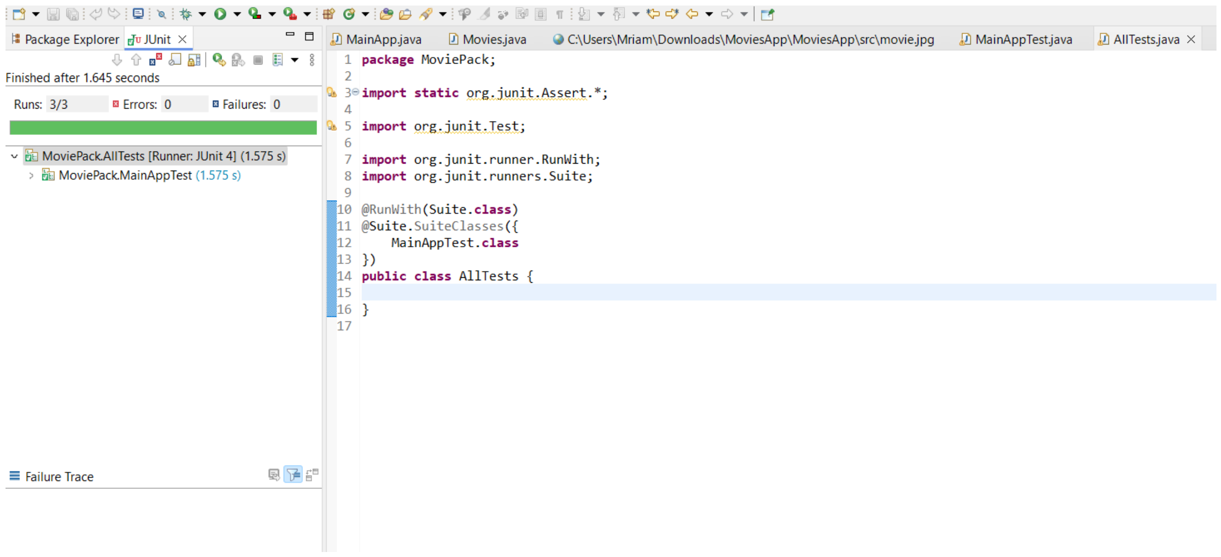
```
inished after 1.657 seconds
Runs: 3/3    Errors: 0    Failures: 0

MoviePack.MainAppTest [Runner: JUnit 4] (1.640 s)
    testSearchMoviesNoSelection (0.402 s)
    testGetSelectedItems (0.095 s)
    testSearchMovies (1.143 s)
```

```java


        // Call the methods to extract selected items
        List<String> selectedGenres = mainApp.getSelectedGenres();
        List<String> selectedDirectors = mainApp.getSelectedDirectors();

        // Assert that the correct items are extracted
        assertEquals(2, selectedGenres.size());
        assertTrue(selectedGenres.contains("Action"));
        assertTrue(selectedGenres.contains("Drama"));

        assertEquals(2, selectedDirectors.size());
        assertTrue(selectedDirectors.contains("Edgar Wright"));
        assertTrue(selectedDirectors.contains("Quentin Tarantino"));


    }
    @Test
    public void testSearchMoviesNoSelection() {
        MainApp mainApp = new MainApp();

        // Simulate no user selection
        List<String> selectedGenres = new ArrayList<>();
        List<String> selectedDirectors = new ArrayList<>();
        List<String> selectedActors = new ArrayList<>();

        // Call the method to search movies with no selection
        mainApp.searchMovies(selectedGenres, selectedDirectors, selectedActors);
        String resultText = mainApp.getResultArea().getText();

        // Assert that the result area displays a message indicating no movies found
        assertEquals("No movies found.", resultText.trim());
    }
```

This test case, **testSearchMoviesNoSelection**, is designed to verify the behavior of the **searchMovies** method in the **MainApp** class when no genres, directors, or actors are selected by the user.

Assertion: The test verifies that the result area of the **MainApp** instance contains the expected text, which should be "No movies found." indicating that no movies matching the empty selection criteria were found.

Test Suite



```
Finished after 1.645 seconds
Runs: 3/3    Errors: 0    Failures: 0

MoviePack.AllTests [Runner: JUnit 4] (1.575 s)
  MoviePack.MainAppTest (1.575 s)
```

```java
package MoviePack;

import static org.junit.Assert.*;

import org.junit.Test;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    MainAppTest.class
})
public class AllTests {

}
```

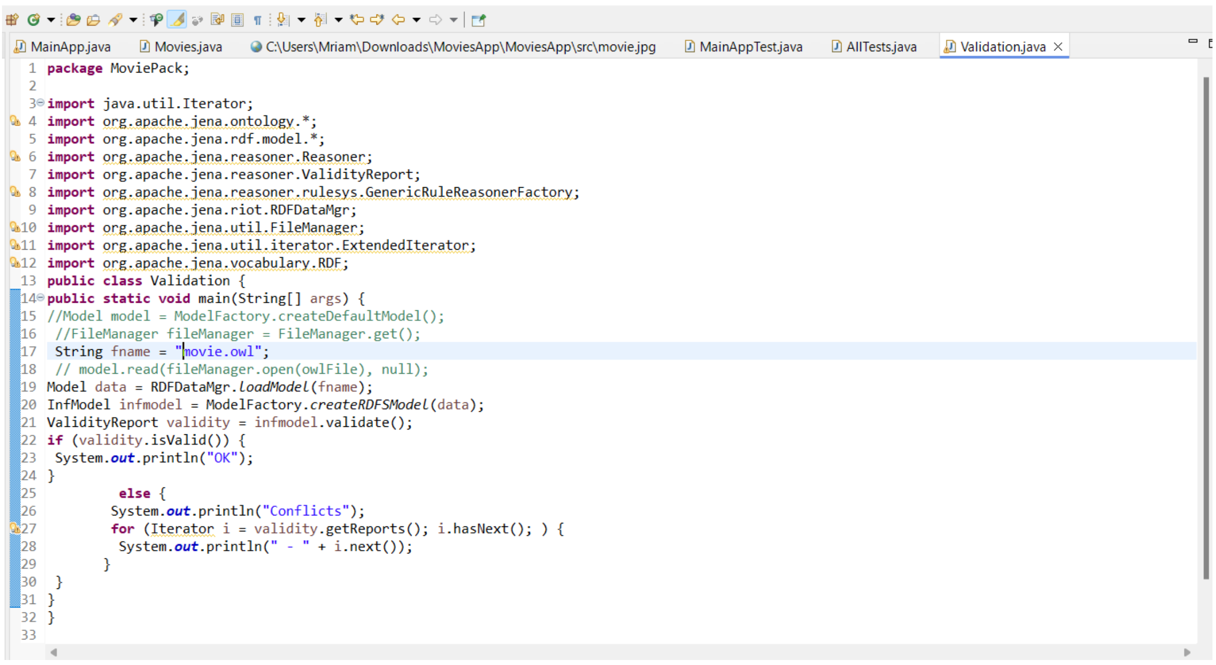
The **AllTests** test suite is a collection of test cases designed to comprehensively evaluate the functionality of the **MainApp** class in the **MoviePack** package. It includes various unit tests that assess different aspects of the **MainApp** class, ensuring its correctness and robustness under different scenarios.

This test suite is to evaluate the correctness and completeness of the ontology, so all the test cases pass.

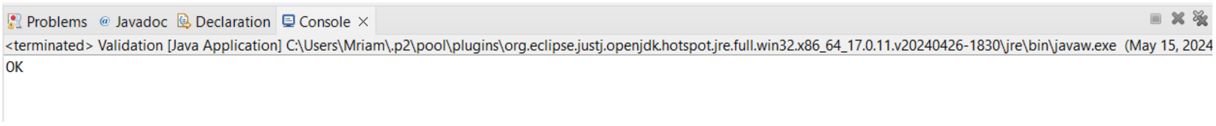
Query Performance

Assessing query performance entails gauging the efficiency and effectiveness of querying operations within the ontology framework. Test cases designed for this aspect aim to evaluate the ontology's responsiveness and scalability across a spectrum of query complexities and sizes. This evaluation encompasses measuring the speed of query execution, analyzing resource utilization metrics such as memory and CPU usage, and scrutinizing the ontology's capacity to manage concurrent or parallel queries. Test scenarios may involve benchmarking query response times, stress-testing the system with a substantial influx of queries, and examining performance metrics under varying load conditions.

Ontology Correctness



```java
package MoviePack;

import java.util.Iterator;
import org.apache.jena.ontology.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ValidityReport;
import org.apache.jena.reasoner.rulesys.GenericRuleReasonerFactory;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.util.FileManager;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.jena.vocabulary.RDF;
public class Validation {
public static void main(String[] args) {
//Model model = ModelFactory.createDefaultModel();
 //FileManager fileManager = FileManager.get();
 String fname = "movie.owl";
 // model.read(fileManager.open(owlFile), null);
Model data = RDFDataMgr.loadModel(fname);
InfModel infmodel = ModelFactory.createRDFSModel(data);
ValidityReport validity = infmodel.validate();
if (validity.isValid()) {
 System.out.println("OK");
}
          else {
         System.out.println("Conflicts");
         for (Iterator i = validity.getReports(); i.hasNext(); ) {
          System.out.println(" - " + i.next());
        }
 }
}
}
```



```
<terminated> Validation [Java Application] C:\Users\Mriam\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe  (May 15, 2024
OK
```

This **Validation** class, is designed to perform validation on an ontology file (**movie.owl**) using the Apache Jena framework. The purpose of this code is to ensure the consistency and correctness of the ontology data. Below is a description of the code's functionality and components:

1. **Import Statements**: The code imports necessary classes from the Apache Jena library to work with ontologies, models, reasoning, and validation.

2. **Main Method**: The entry point of the program. It loads the ontology file (**movie.owl**) into a Jena **Model** using **RDFDataMgr.loadModel**.

3. **InfModel Creation**: An inference model (**InfModel**) is created from the loaded data model (**Model**). Inference models are used for reasoning over RDF data, applying schema information and inferencing rules.

4. **Validation**: The **validate()** method is called on the inference model (**infmodel**) to perform validation. The **validate()** method checks for consistency and correctness in the ontology data.

5. **Validity Report**: The result of the validation is stored in a **ValidityReport** object (**validity**). If the ontology is valid (i.e., no conflicts or inconsistencies are found), it prints "OK" to the console. Otherwise, it prints "Conflicts" followed by the details of the validation reports.