



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

Problem Description:.....	2
Part I: Modeling the Ontology	3
Classes:	3
Data Properties:	4
Object Properties:	4
Constraints:	5
Part II: Populating the Ontology.....	7
Individuals:	7
Ontology Graph:	9
Movie Instances.....	9
Person Instances.....	10
Genre Instances.....	10
Part III: Querying the Ontology	11
Write SPARQL queries to response to the following:.....	11
Propose 10 SPARQL queries:	21
Part IV: Manipulating the ontology using Jena	31
Part V: Java application	37
Testing : Test Cases	41
Test case 1	41
Test case 2	42
Test case 3	43
Test Suite	44

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.

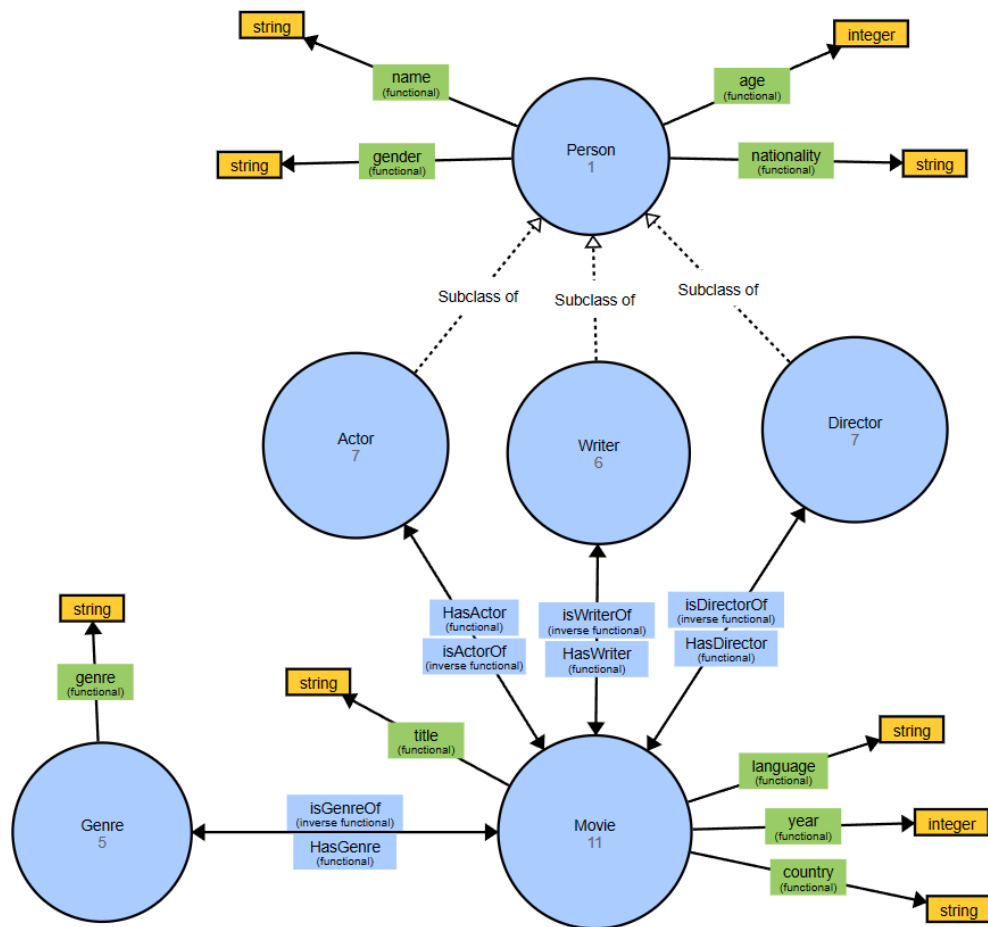
Key Objectives:

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.

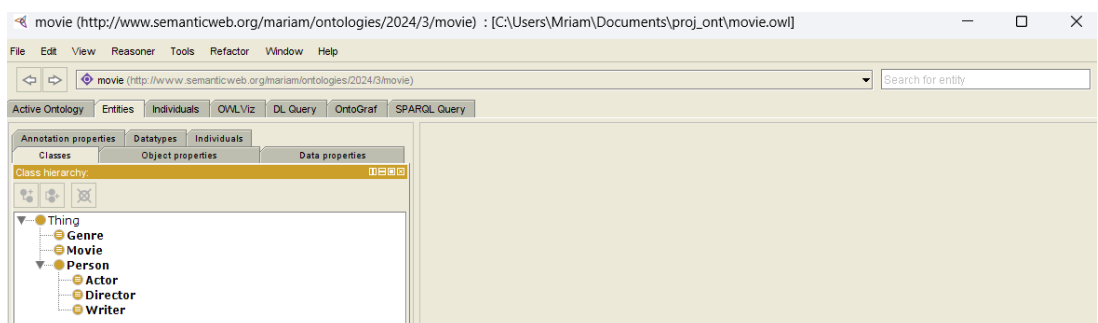


Part I: Modeling the Ontology



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.



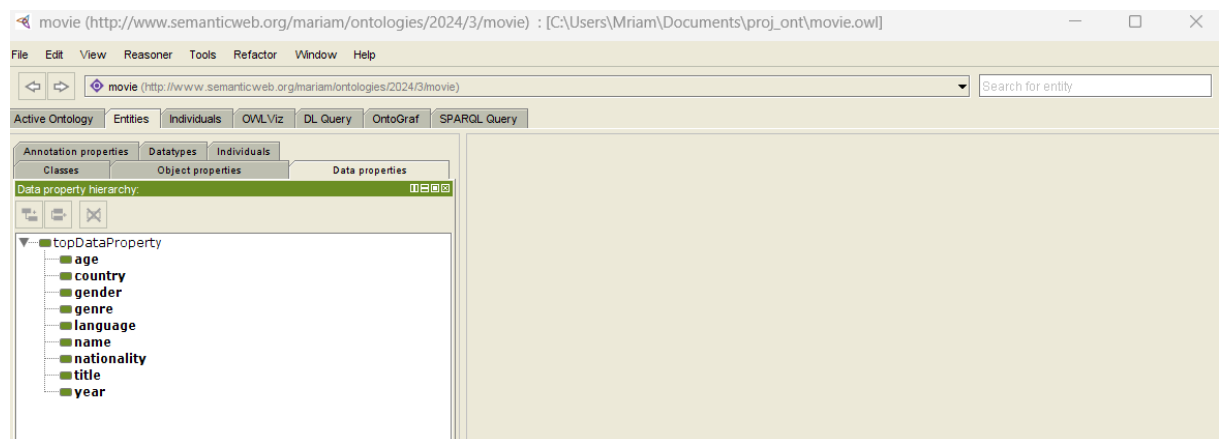
Data Properties:

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.

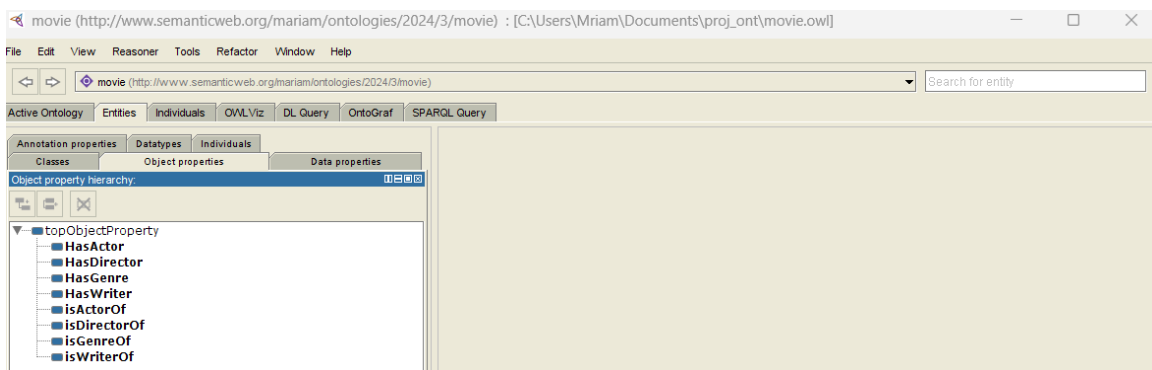
Indicates the country, language, title, genre, and year of release of a movie.



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.




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

Sub Class Of 
● Person

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

Equivalent To 
● isDirectorOf min 1 Movie

Sub Class Of 
● Person


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

Equivalent To 
● isWriterOf min 1 Movie


Sub Class 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]

Domains (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

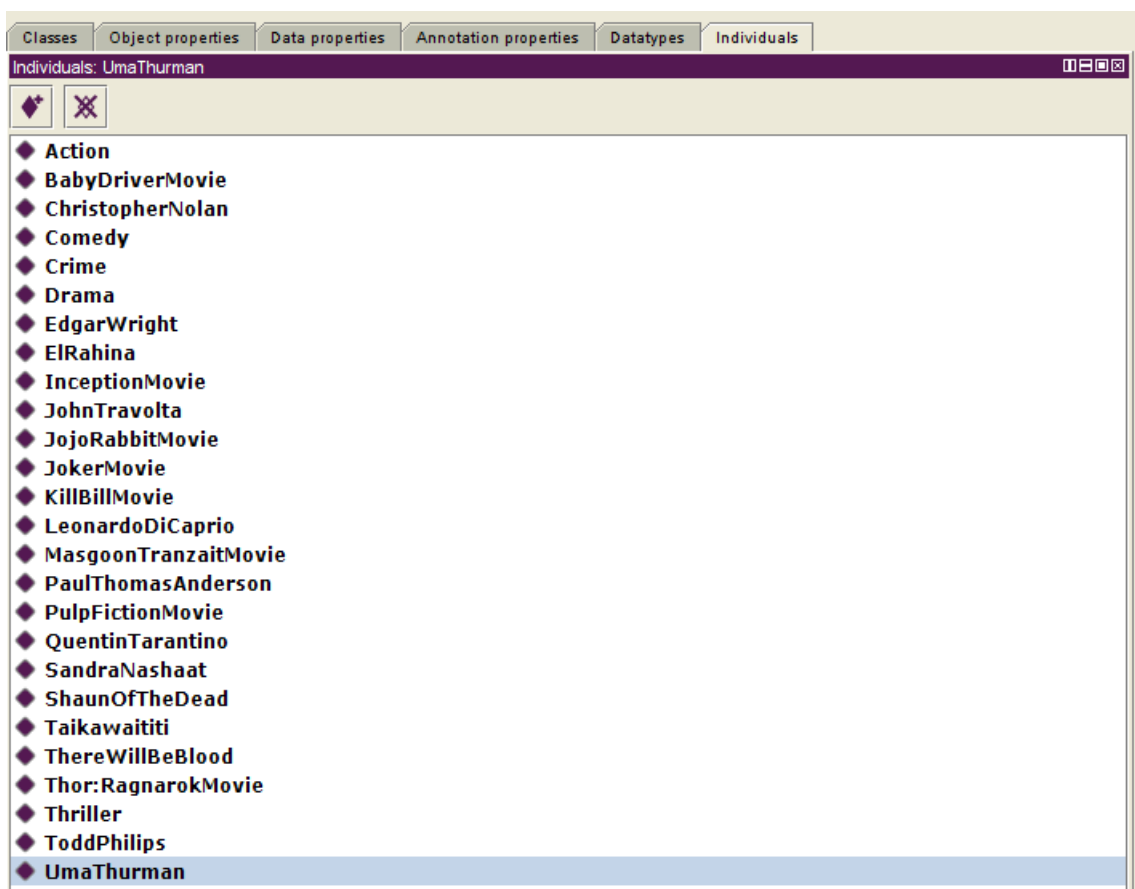
Individuals:

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:Ragnarok, 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.

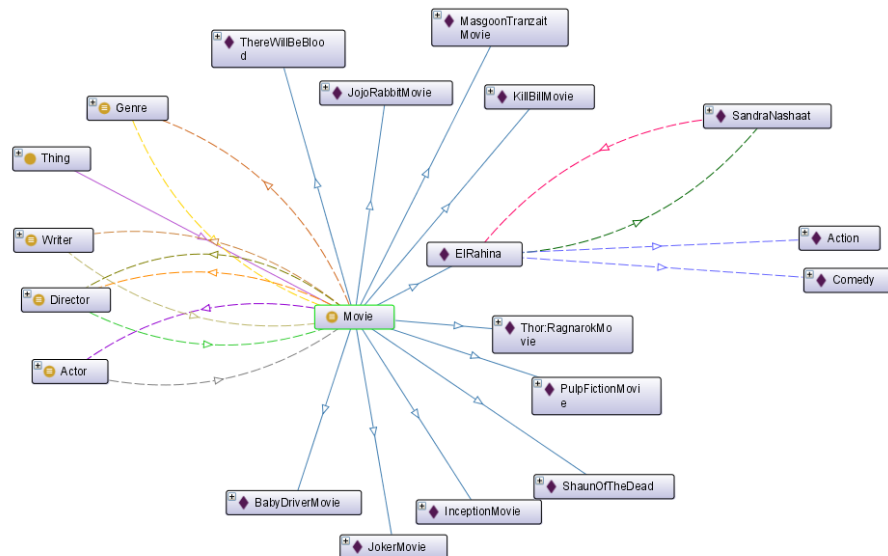
- h) Paul Thomas Anderson, an American filmmaker, aged 51, known for writing and directing "There Will Be Blood" He often includes himself in small acting roles within his films.
- i) Sandra Nashaat, Female Egyptian Director, 54 years old, director of elRahina and Masgoon Tranzait.
- j) Todd Philips , American , 51 years old, director and writer of the Joker Movie.



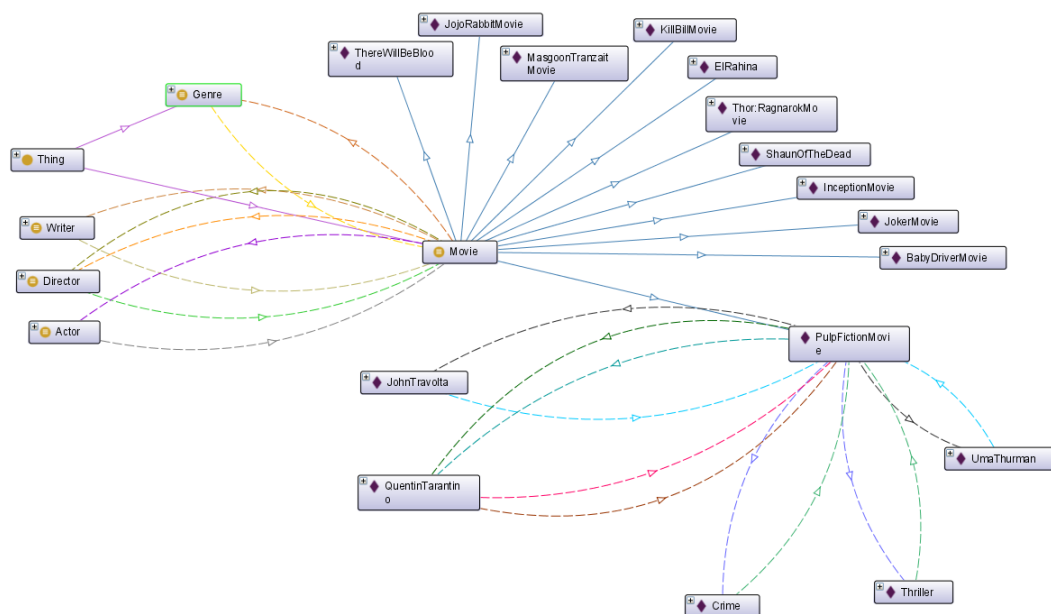
Ontology Graph:

Movie Instances

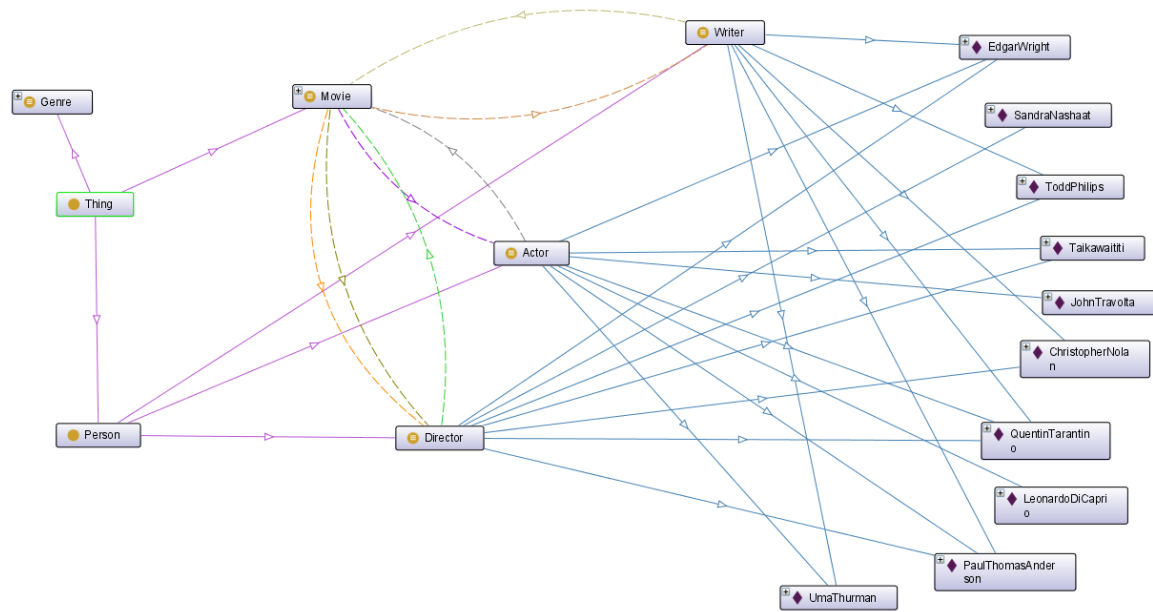
- Example **El Rahina** Movie



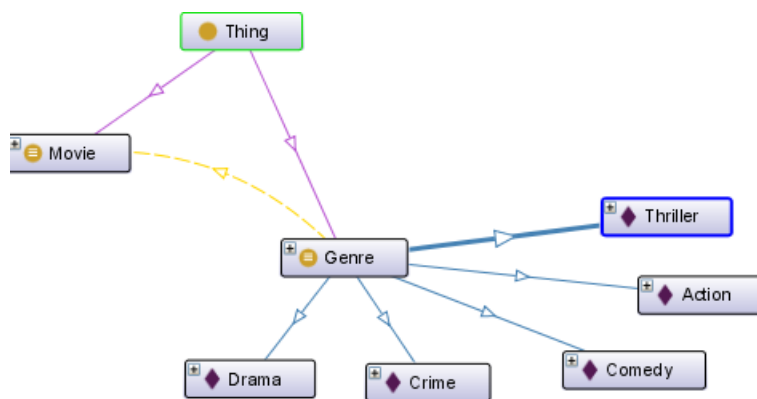
- Example **Pulp Fiction** Movie



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#>

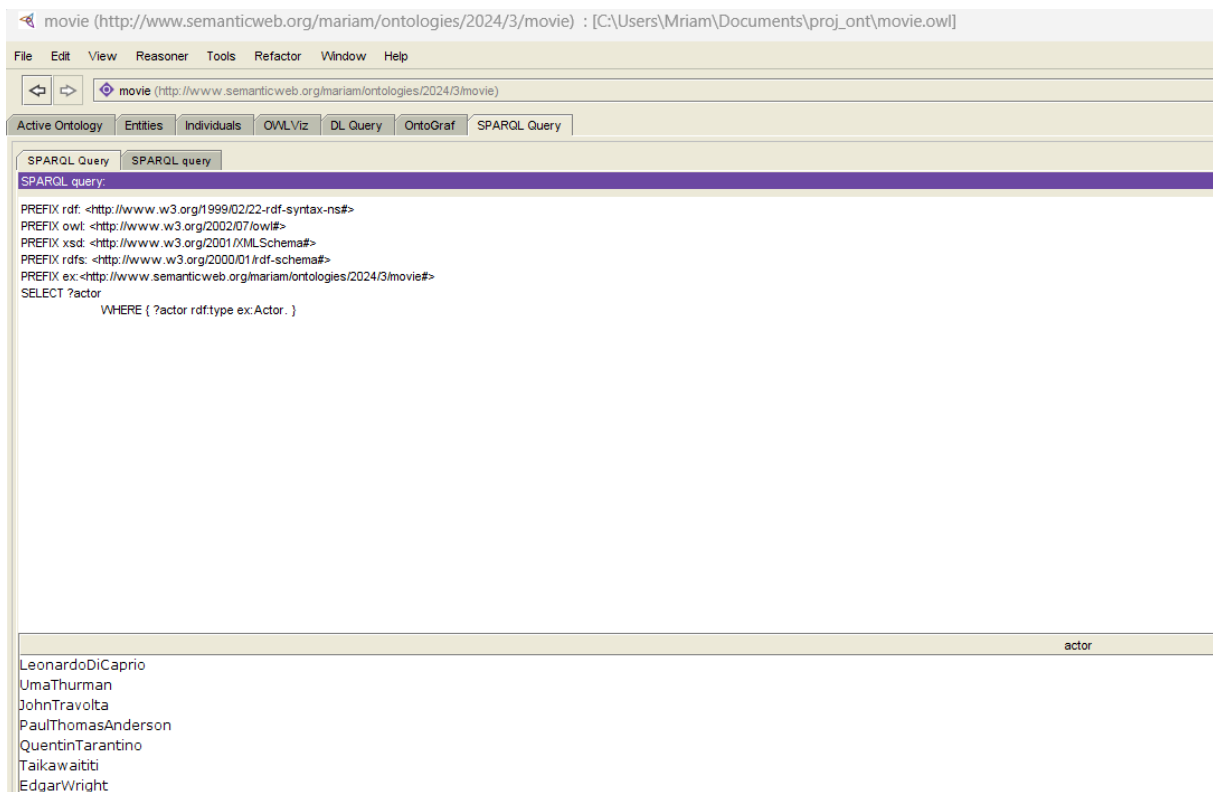
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

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. }



The screenshot shows a web-based SPARQL query interface. The address bar displays the URL: `movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]`. The interface includes a menu bar with options: File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. Below the menu is a toolbar with navigation icons and a text input field containing the same URL. The main workspace is divided into two tabs: "Active Ontology" and "SPARQL Query". The "SPARQL Query" tab is active, showing the following 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. }
```

Below the query, the results are displayed in a table with a single column header "actor". The results list the following names:

actor
LeonardoDiCaprio
UmaThurman
JohnTravolta
PaulThomasAnderson
QuentinTarantino
Taikawaibiti
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 OML Viz 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#>

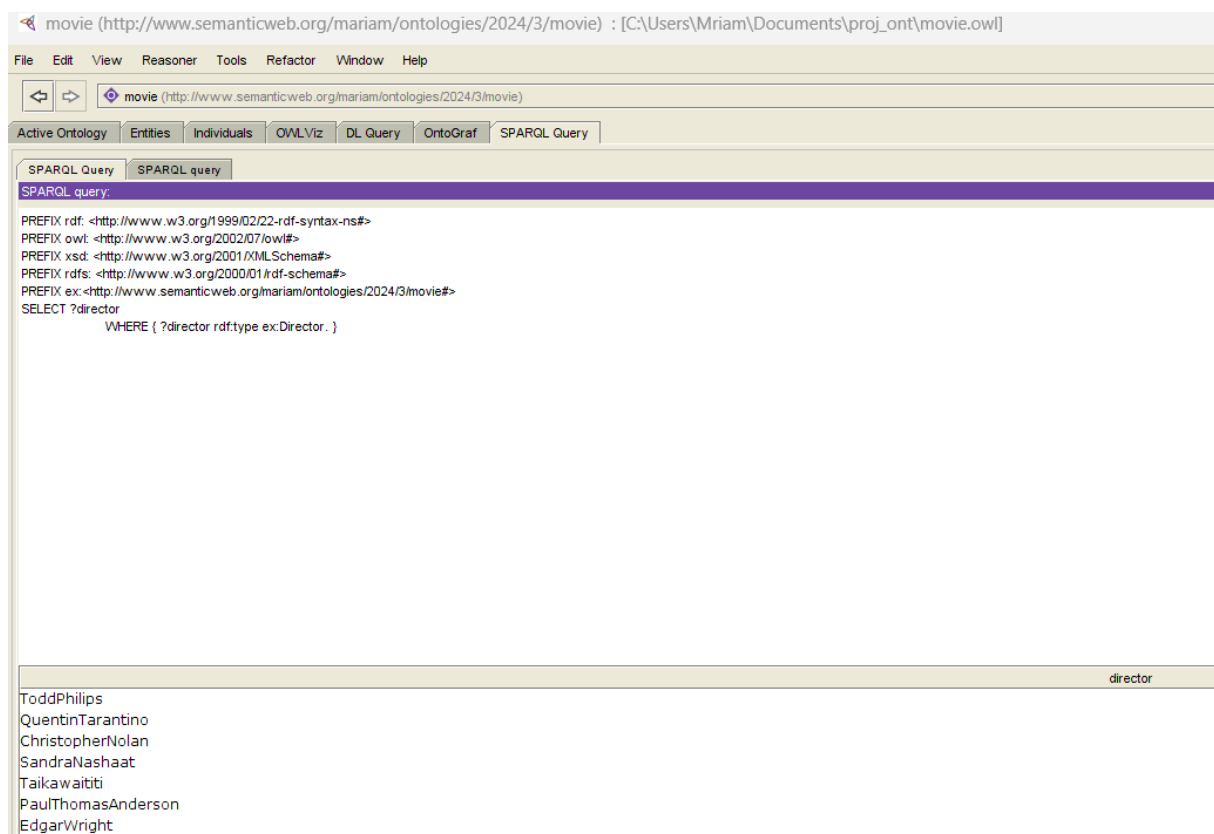
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

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. }



The screenshot shows a web-based SPARQL query interface. The browser address bar displays the URL: `movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]`. The interface includes a menu bar (File, Edit, View, Reasoner, Tools, Refactor, Window, Help) and a toolbar with navigation icons. Below the toolbar, there are tabs for 'Active Ontology', 'Entities', 'Individuals', 'OWL Viz', 'DL Query', 'OntoGraf', and 'SPARQL Query'. The 'SPARQL Query' tab is active, showing a text area with the following 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. }
```

Below the query text area, there is a table displaying the results. The table has a single column header 'director' and lists the following names as results:

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.

}

The screenshot shows a web browser window with the URL <http://www.semanticweb.org/mariam/ontologies/2024/3/movie>. The browser has a menu bar with File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. Below the menu bar is a search bar with the text "movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)" and a "Search for entity" button. The main content area has tabs for Active Ontology, Entities, Individuals, OWL Viz, DL Query, OntoGraf, and SPARQL Query. The SPARQL Query tab is active, showing the following 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 ?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.
}
```

Below the query, there is a table with two columns: movieName and directorName. The table contains three rows of results:

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

At the bottom of the window, there is an "Execute" button.



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 {

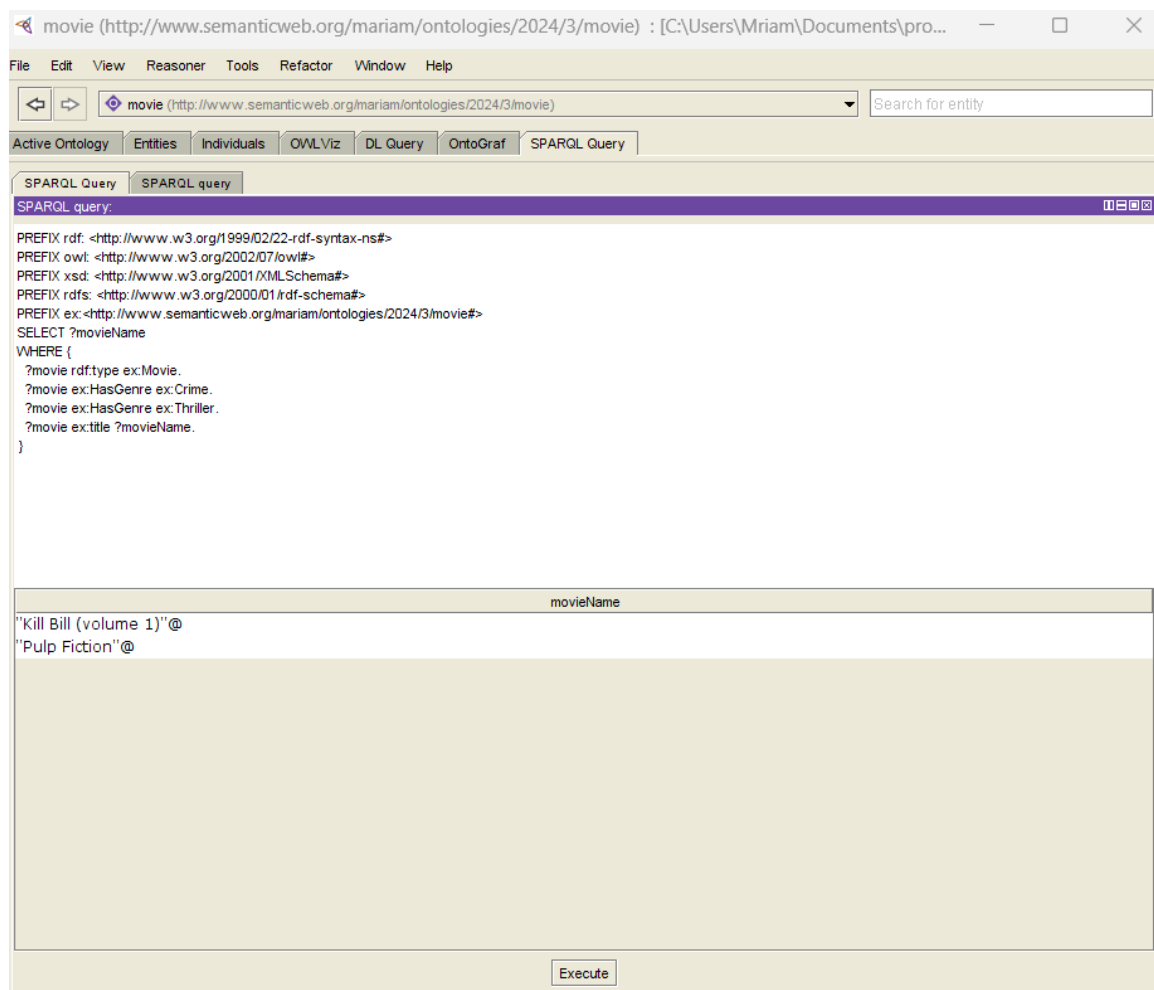
?movie rdf:type ex:Movie.

?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:

```
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"@"

Execute



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.

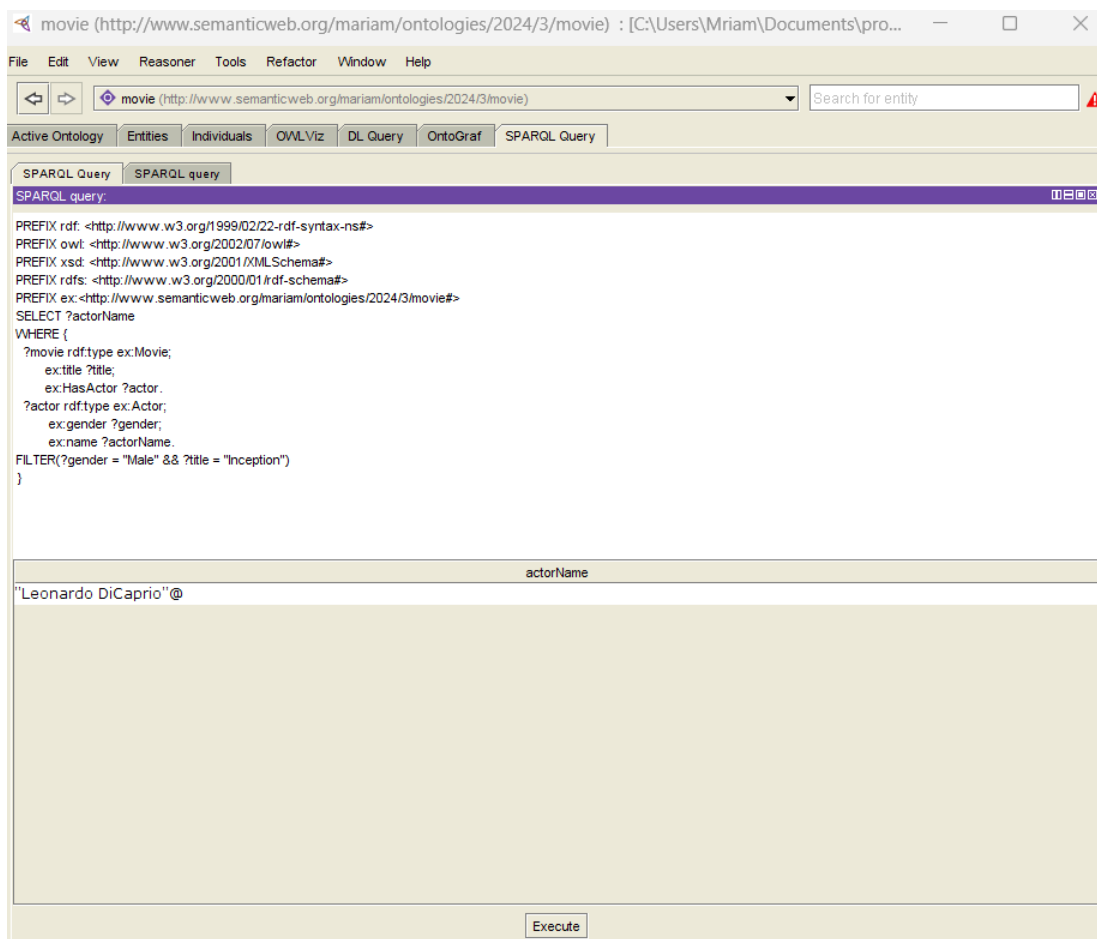
?actor rdf:type ex:Actor;

ex:gender ?gender;

ex:name ?actorName.

FILTER(?gender = "Male" && ?title = "Inception")

}



The screenshot shows a web application window titled "movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...". The application has a menu bar with "File", "Edit", "View", "Reasoner", "Tools", "Refactor", "Window", and "Help". Below the menu bar is a toolbar with a search icon, a dropdown menu showing the current ontology, and a search input field labeled "Search for entity". The main interface has tabs for "Active Ontology", "Entities", "Individuals", "OWL Viz", "DL Query", "OntoGraf", and "SPARQL Query". The "SPARQL Query" tab is active, showing a text area with the following 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 ?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")
}
```

Below the query text area, there is a table with the column header "actorName". The table contains one row with the value "Leonardo DiCaprio"@. At the bottom of the window, there is an "Execute" button.



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.

}

The screenshot shows a web browser window titled "movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...". The browser has a menu bar with File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. Below the menu bar is a search bar with the text "movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)" and a "Search for entity" button. Below the search bar are tabs for Active Ontology, Entities, Individuals, OWL Viz, DL Query, OntoGraf, and SPARQL Query. The SPARQL Query tab is active, showing a text area with the following 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 (COUNT(?movie) AS ?count)
WHERE {
  ?movie rdf:type ex:Movie.
  ?movie ex:HasGenre ex:Action.
  ?movie ex:HasGenre ex:Thriller.
}
```

Below the query text area is a table with one column labeled "count". The table contains one row with the value "1". Below the table is an "Execute" button.



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.

}

The screenshot shows a web application interface for a SPARQL query engine. The browser window title is "movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\pro...". The application has a menu bar (File, Edit, View, Reasoner, Tools, Refactor, Window, Help) and a toolbar with navigation and search icons. Below the toolbar is a tabbed interface with "Active Ontology", "Entities", "Individuals", "OWL Viz", "DL Query", "OntoGraf", and "SPARQL Query" selected. The "SPARQL Query" tab is active, showing a text area with the following 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
WHERE {
  ?movie rdf:type ex:Movie;
    ex:HasWriter ex:EdgarWright;
    ex:title ?movieTitle.
}
```

Below the query editor is a results table with the header "movieTitle". It contains two rows of results:

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

At the bottom of the interface is an "Execute" button.



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

actorName
"John Travolta"@
"Quentin Tarantino"@

Execute



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:

```
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"@

Execute



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"@
"Tika Waijib"@	"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")}

}

The screenshot shows a SPARQL query editor window titled "movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie)". The query is as follows:

```
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")
  }
}
```

The results table shows the following data:

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 .

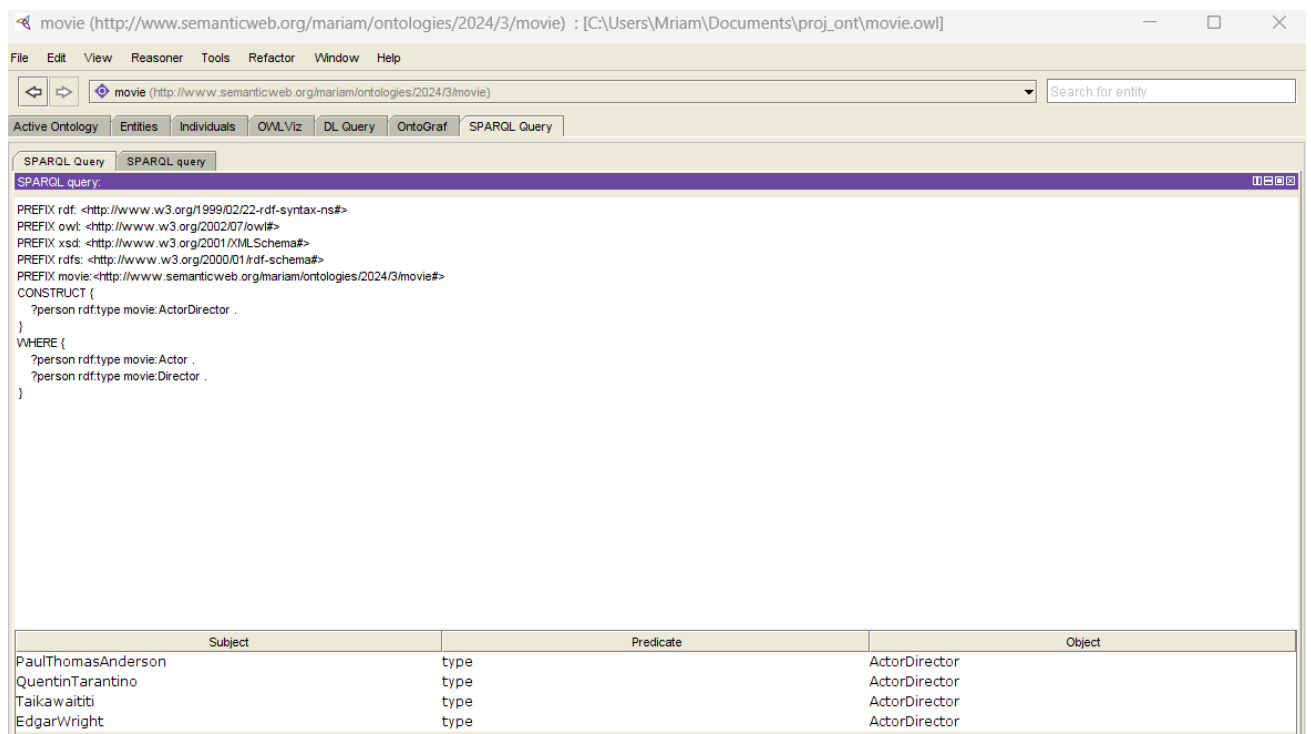
}

WHERE {

 ?person rdf:type movie:Actor .

 ?person rdf:type movie:Director .

}



The screenshot shows a web application interface for a SPARQL query. The title bar indicates the file path: `movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Miriam\Documents\proj_ont\movie.owl]`. The interface includes a menu bar (File, Edit, View, Reasoner, Tools, Refactor, Window, Help) and a toolbar with navigation icons. Below the toolbar is a search bar labeled "Search for entity". The main area is divided into tabs: "Active Ontology", "Entities", "Individuals", "OWL Viz", "DL Query", "OntoGraf", and "SPARQL Query". The "SPARQL Query" tab is active, displaying the following 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 .
}
```

Below the query editor, a table displays the results of the query. The table has four columns: "Subject", "Predicate", and "Object". The results show four individuals: PaulThomasAnderson, QuentinTarantino, Taikawaititi, and EdgarWright, all with the predicate "type" and the object "ActorDirector".

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 {

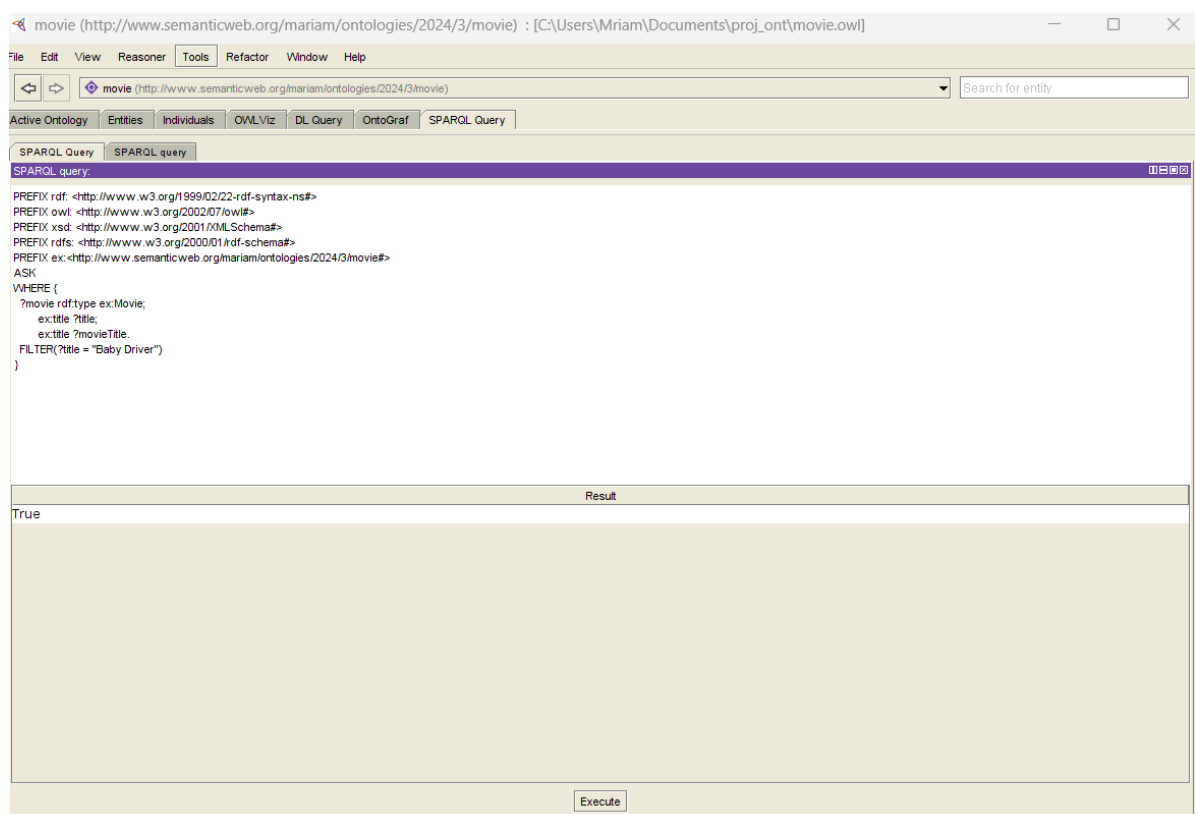
?movie rdf:type ex:Movie;

ex:title ?title;

ex:title ?movieTitle.

FILTER(?title = "Baby Driver")

}



The screenshot shows a web-based SPARQL query editor. The title bar indicates the file path: `movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\Users\Mriam\Documents\proj_ont\movie.owl]`. The interface includes a menu bar (File, Edit, View, Reasoner, Tools, Refactor, Window, Help) and a toolbar with navigation and search icons. Below the toolbar is a tabbed interface with 'Active Ontology', 'Entities', 'Individuals', 'OWL Viz', 'DL Query', 'OntoGraf', and 'SPARQL Query'. The 'SPARQL Query' tab is active, displaying the query text:

```
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")
}
```

 Below the query editor is a 'Result' section showing the output:

```
True
```

 At the bottom right of the window is an 'Execute' button.



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

The screenshot shows a web browser window with a SPARQL query interface. The query is: `DESCRIBE ex:Director`. The results are displayed in a table with three columns: Subject, Predicate, and Object.

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
Taikawaibiti	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 .

}

The screenshot shows a web browser window with the URL `http://www.semanticweb.org/mariam/ontologies/2024/3/movie`. The interface includes a menu bar (File, Edit, View, Reasoner, Tools, Refactor, Window, Help) and a toolbar with navigation and search icons. Below the toolbar, there are tabs for "Data Properties", "Annotation Properties", "Individuals", "OWL Viz", "DL Query", "OntoGraf", "Ontology Differences", and "SPARQL Query". The "SPARQL Query" tab is active, displaying the query text. Below the query, there is a table with two columns: "movieTitle" and "genre". The table contains five rows of results. At the bottom of the interface, there is an "Execute" button and a status bar that reads "No Reasoner set. Select a reasoner from the Reasoner menu" and "Show Inferences" (checked).

```
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 ?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

Execute

No Reasoner set. Select a reasoner from the Reasoner menu ☒ Show Inferences



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)

}

The screenshot shows a web application window titled "movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\semester...". The interface includes a menu bar (File, Edit, View, Reasoner, Tools, Refactor, Window, Help) and a toolbar with navigation icons and a search bar. Below the toolbar is a tabbed interface with tabs for "Data Properties", "Annotation Properties", "Individuals", "OWL Viz", "DL Query", "OntoGraf", "Ontology Differences", "SPARQL Query", "Active Ontology", "Entities", "Classes", and "Object Properties". The "SPARQL Query" tab is active, displaying the following 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 {
  ?movie rdf:type ex:Movie .
  ?movie ex:HasActor ?actor .
  ?movie ex:year ?year .
  FILTER (?year < 2010)
}
```

Below the query editor, the results are displayed under the heading "actor". The results list the following names:

- PaulThomasAnderson
- EdgarWright
- UmaThurman
- QuentinTarantino
- JohnTravolta

At the bottom of the interface, there is an "Execute" button and a status bar that reads "No Reasoner set. Select a reasoner from the Reasoner menu" with a checked box for "Show Inferences".

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 .

}

The screenshot shows a web application window titled "movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\semester...". The interface includes a menu bar (File, Edit, View, Reasoner, Tools, Refactor, Window, Help) and a toolbar with navigation and search icons. Below the toolbar is a tabbed interface with tabs for "Data Properties", "Annotation Properties", "Individuals", "OWL Viz", "DL Query", "OntoGraf", "Ontology Differences", "SPARQL Query", and "Object Properties". The "SPARQL Query" tab is active, displaying the following 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 ?nationality
WHERE {
  ex:PulpFictionMovie rdf:type ex:Movie .
  ex:PulpFictionMovie ex:HasDirector ?director .
  ?director ex:nationality ?nationality .
}
```

Below the query, the results are displayed in a table with the column header "nationality". The first row contains the value "American"@.

At the bottom of the interface, there is an "Execute" button and a status bar that reads "No Reasoner set. Select a reasoner from the Reasoner menu" with a checked box for "Show Inferences".

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

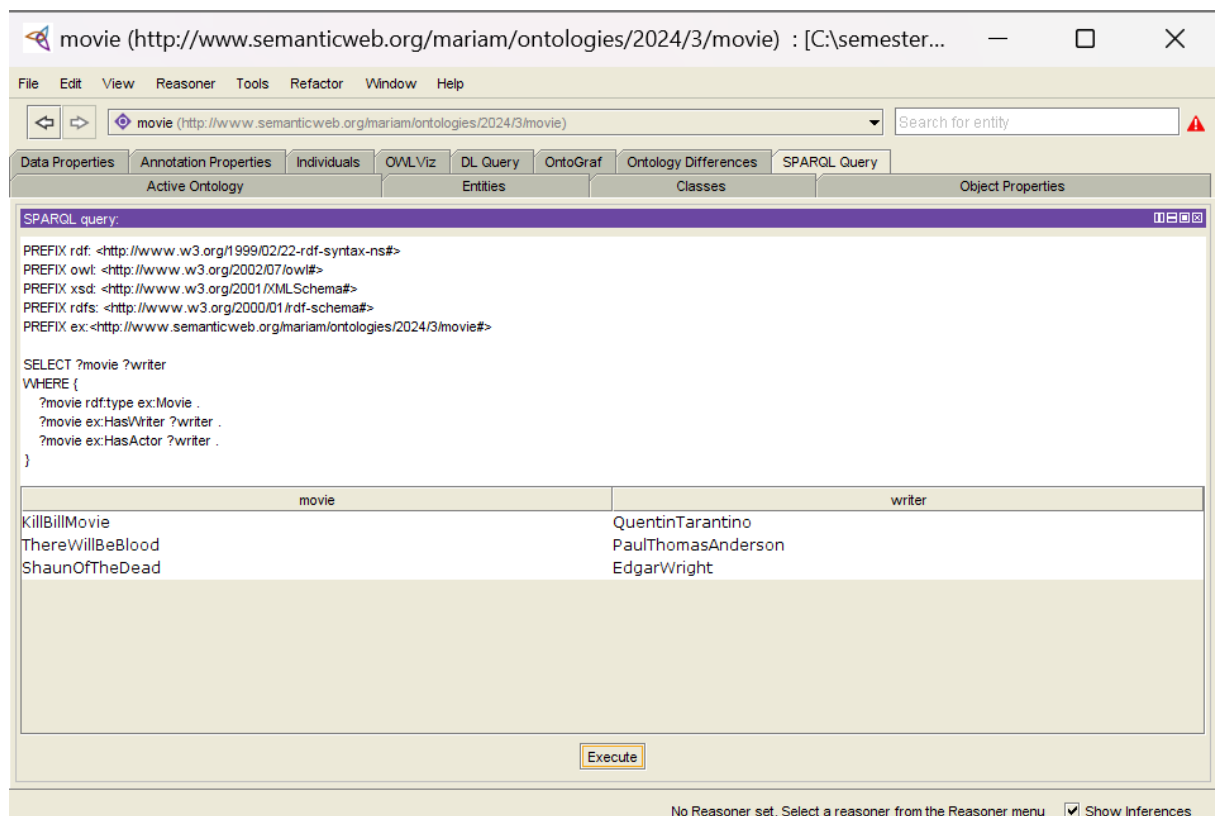
WHERE {

?movie rdf:type ex:Movie .

?movie ex:HasWriter ?writer .

?movie ex:HasActor ?writer .

}



The screenshot shows a web application window titled "movie (http://www.semanticweb.org/mariam/ontologies/2024/3/movie) : [C:\semester...". The interface includes a menu bar (File, Edit, View, Reasoner, Tools, Refactor, Window, Help) and a toolbar with navigation and search icons. Below the toolbar is a tabbed interface with tabs for "Data Properties", "Annotation Properties", "Individuals", "OWL Viz", "DL Query", "OntoGraf", "Ontology Differences", "SPARQL Query", and "Object Properties". The "SPARQL Query" tab is active, displaying the following 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 ?movie ?writer
WHERE {
  ?movie rdf:type ex:Movie .
  ?movie ex:HasWriter ?writer .
  ?movie ex:HasActor ?writer .
}
```

Below the query, the results are displayed in a table with two columns: "movie" and "writer".

movie	writer
KillBillMovie	QuentinTarantino
ThereWillBeBlood	PaulThomasAnderson
ShaunOfTheDead	EdgarWright

At the bottom of the table, there is an "Execute" button. The status bar at the bottom of the window indicates "No Reasoner set. Select a reasoner from the Reasoner menu" and has a checkbox for "Show Inferences" which is checked.

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") .

}

The screenshot shows a web browser window with the URL <http://www.semanticweb.org/mariam/ontologies/2024/3/movie>. The browser has a menu bar (File, Edit, View, Reasoner, Tools, Refactor, Window, Help) and a toolbar with navigation and search icons. Below the toolbar is a tabbed interface with 'Data Properties', 'Annotation Properties', 'Individuals', 'OWL Viz', 'DL Query', 'OntoGraf', 'Ontology Differences', 'SPARQL Query', and 'Object Properties'. The 'SPARQL Query' tab is active, displaying the following 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
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") .
}
```

Below the query, there is a table with two columns: 'movieTitle' and 'directorName'. The table contains the following data:

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"@

At the bottom of the window, there is a status bar that says 'No Reasoner set. Select a reasoner from the Reasoner menu' and a checkbox for 'Show Inferences' which is checked.

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 ×
14 public class jena1 {
15     public static void main(String[] args) {
16         // Load the ontology
17         OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
18         FileManager fileManager = FileManager.get();
19         String owlFile = "Data/movie.owl";
20         model.read(fileManager.open(owlFile), null);
21
22         // List Persons
23         OntClass personClass = model.getOntClass("http://www.semanticweb.org/mariam/ontologies/2024/3/mc
24         ExtendedIterator<? extends OntResource> personIterator = personClass.listInstances();
25         while (personIterator.hasNext()) {
26             OntResource personResource = personIterator.next();
27
28             // Get the name of the person
29             Property nameProperty = model.getProperty("http://www.semanticweb.org/mariam/ontologies/2024
30             StmtIterator personNameIterator = personResource.listProperties(nameProperty);
31             while (personNameIterator.hasNext()) {
32                 Statement personNameStatement = personNameIterator.next();
33                 RDFNode personNameNode = personNameStatement.getObject();
34
35             }
36         }
37     }
38 }
```

Problems Javadoc Declaration Console ×

```
<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 Phillips
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 ×
16
17 public class jena2 {
18     public static void main(String[] args) {
19         // Load the ontology
20         OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
21         FileManager fileManager = FileManager.get();
22         String owlFile = "Data/movie.owl";
23         model.read(fileManager.open(owlFile), null);
24
25         // Load the SPARQL query from file
26         String queryFile = "Data/personsQuery.txt";
27         String queryString = readQueryFromFile(queryFile);
28
29         // Execute the query
30         Query query = QueryFactory.create(queryString);
31         try (QueryExecution qexec = QueryExecutionFactory.create(query, model)) {
32             ResultSet results = qexec.execSelect();
33             while (results.hasNext()) {
34                 QuerySolution solution = results.next();
35                 String personName = solution.getLiteral("name").getString();
36             }
37         }
38     }
39 }
```

Problems Javadoc Declaration Console ×

<terminated> jena2 [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justi.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:

```
jena2.java personsQuery.txt ×
1 PREFIX owl: <http://www.w3.org/2002/07/owl#>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX movies: <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
5
6 SELECT ?name
7 WHERE {
8     ?person rdf:type movies:Person .
9     ?person movies:name ?name .
10 }
11
```



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

```
jena3.java x
15
16 public class jena3 {
17     public static void main(String[] args) {
18         // Load the ontology
19         OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
20         FileManager fileManager = FileManager.get();
21         String owlFile = "Data/movie.owl"; // Replace with the actual filename
22         model.read(fileManager.open(owlFile), null);
23
24         // Create a reasoner with OWL rules
25         Reasoner reasoner = GenericRuleReasonerFactory.theInstance().create(null);
26         InfModel infModel = ModelFactory.createInfModel(reasoner, model);
27
28         // List all actors
29         OntClass actorClass = model.getOntClass("http://www.semanticweb.org/mariam/ontologies/2024/3/mov");
30         ExtendedIterator<? extends OntResource> actorIterator = actorClass.listInstances();
31         while (actorIterator.hasNext()) {
32             OntResource actorResource = actorIterator.next();
33             Property nameProperty = model.getProperty("http://www.semanticweb.org/mariam/ontologies/2024/3/mov#name");
34             StmtIterator actorNameIterator = actorResource.listProperties(nameProperty);
35             while (actorNameIterator.hasNext()) {
36                 Stmt stmt = actorNameIterator.next();
37                 String actorName = stmt.getObject().toString();
38                 System.out.println("Actor: " + actorName);
39             }
40         }
41     }
42 }
```

Problems Javadoc Declaration Console x

```
<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:

- Reads a name of a movie
- If it doesn't exist displays an error message
- Else, display its year, country, genres and actors

Movie not found:

```

10
11 public class jena4 {
12     public static void main(String[] args) {
13         // Load the ontology
14         OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
15         FileManager fileManager = FileManager.get();
16         String owlFile = "Data/movie.owl"; // Replace with the actual filename
17         model.read(fileManager.open(owlFile), null);
18
19         // Read the title of a movie (replace with your desired movie title)
20         String movieTitle = "Joker";
21         // Create a Scanner object to read user input
22         Scanner scanner = new Scanner(System.in);
23
24         // Prompt the user to enter the movie title
25         System.out.print("Enter the movie title: ");
26         String movieTitle = scanner.nextLine();
27
28         // Find the movie with the given title
29         OntClass movieClass = model.getOntClass("http://www.semanticweb.org/mariam/ontology#Joker");
30
31         // Print the result
32         System.out.println("Movie found: " + movieClass.getName());
33     }
34 }

```

Problems Javadoc Declaration Console ×

<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:

```
jena4.java x
11 public class jena4 {
12     public static void main(String[] args) {
13         // Load the ontology
14         OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
15         FileManager fileManager = FileManager.get();
16         String owlFile = "Data/movie.owl"; // Replace with the actual filename
17         model.read(fileManager.open(owlFile), null);
18
19         // Read the title of a movie (replace with your desired movie title)
20         String movieTitle = "Joker";
21         // Create a Scanner object to read user input
22         Scanner scanner = new Scanner(System.in);
23
24         // Prompt the user to enter the movie title
25         System.out.print("Enter the movie title: ");
26         String movieTitle = scanner.nextLine();
27
28         // Find the movie with the given title
29         OntClass movieClass = model.getOntClass("http://www.semanticweb.org/mariam/ontology#Movie");
30         List<Resource> movies = movieClass.listInstances();
    }
}
```

Problems Javadoc Declaration Console x

<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.

```
jena4.java  jena5.java ×
8
9 public class jena5 {
10     public static void main(String[] args) {
11         // Load the ontology
12         Model model = ModelFactory.createDefaultModel();
13         FileManager fileManager = FileManager.get();
14         String owlFile = "Data/movie.owl";
15         model.read(fileManager.open(owlFile), null);
16
17         // Read the SPARQL query from file
18         String queryFile = "Data/movieRules.txt";
19         String queryString = readQueryFromFile(queryFile);
20
21         // Execute the query and create a new model with the constructed triples
22         Model resultModel = ModelFactory.createDefaultModel();
23         try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
24             resultModel = qexec.execConstruct();
25         }
26
27         // Extract and display the names of the persons who are both actors and directors

```

```
Problems  Javadoc  Declaration  Console ×
<terminated> jena5 [Java Application] C:\Users\adham\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1
Persons who are both actors and directors:
Taika Waititi
Quentin Tarantino
Paul Thomas Anderson
Edgar Wright
```

jena 5

jena5 Rule:

```
jena4.java  jena5.java  movieRules.txt ×
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX movies: <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#>
3
4 CONSTRUCT {
5     ?person movies:name ?name .
6 }
7 WHERE {
8     ?person rdf:type movies:Actor .
9     ?person rdf:type movies:Director .
10    ?person movies:name ?name
11 }
12
```


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

```
ask_before_2... ask_movies... MainApp.java Movies.java ask_directo... jena1.java C:\semester... jena6.java x
6 import org.apache.jena.rdf.model.ModelFactory;
7 import org.apache.jena.util.FileManager;
8 import org.apache.jena.reasoner.Reasoner;
9 import org.apache.jena.reasoner.rulesys.*;
10 import org.apache.jena.vocabulary.RDF;
11
12 public class jena6 {
13     public static void main(String[] args) {
14         // Load the ontology
15         Model model = ModelFactory.createDefaultModel();
16         FileManager fileManager = FileManager.get();
17         String owlFile = "Data/movie.owl"; // Replace with the actual filename
18         model.read(fileManager.open(owlFile), null);
19
20         // Define rules
21         String rule1 = "[ruleHasTwoRoles: (?movie <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#Has";
22         String rule2 = "[ruleMoviesDirectedByQT: (?movie <http://www.semanticweb.org/mariam/ontologies/2024/3/mo";
23         String rule3 = "[ruleMoviesInUSA: (?movie <http://www.semanticweb.org/mariam/ontologies/2024/3/movie#cou";
24
25         // Create a reasoner with the rules
26         Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rule1 + " " + rule2 + " " + rule3));
27
28         // Apply the reasoner to the model
29         InfModel infModel = ModelFactory.createInfModel(reasoner, model);
30
31         // Execute queries
32
33         askForTwoRoles(infModel);
34         askMoviesDirectedByQuentinTarantino(infModel);
35         askMoviesInCountry(infModel, "USA");
36     }
37 }
```

jena 6

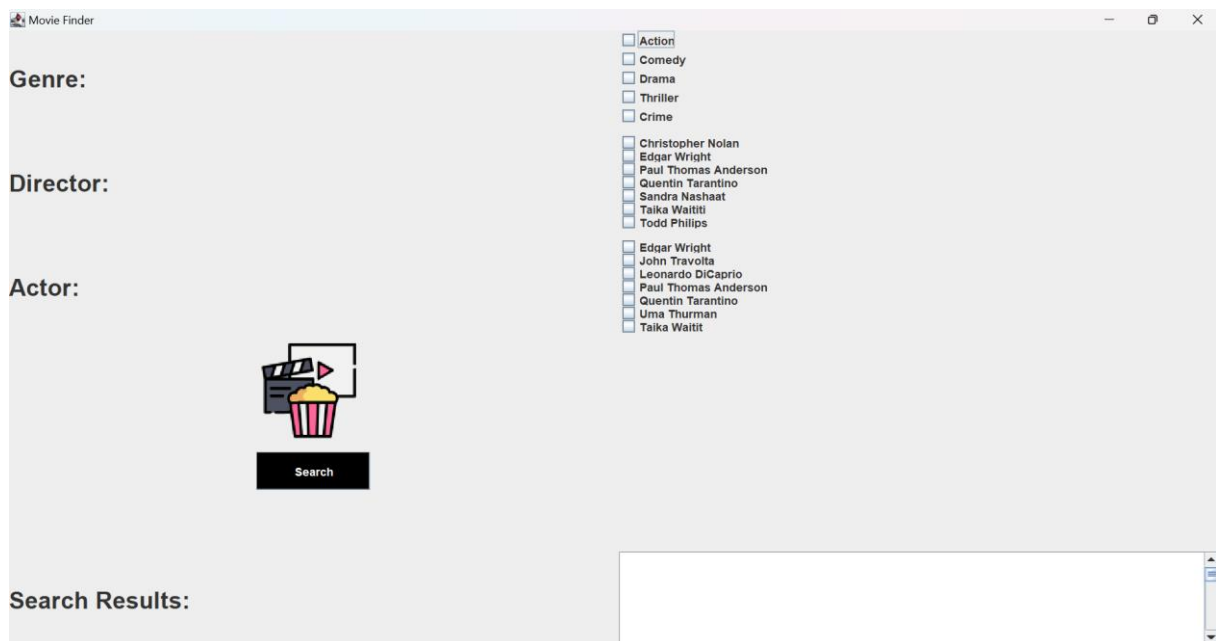
jena6 full Output:

```
Problems Javadoc Declaration Console x
<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:

The screenshot shows the 'Movie Finder' application window. It has a search form with fields for 'Genre:', 'Director:', and 'Actor:'. To the right of these fields is a list of checkboxes for genres (Action, Comedy, Drama, Thriller, Crime) and directors (Christopher Nolan, Edgar Wright, Paul Thomas Anderson, Quentin Tarantino, Sandra Nashaat, Taika Waititi, Todd Phillips). Below the search fields is a 'Search' button. The 'Search Results:' section at the bottom displays a list of movies. The first three results are: 'Movie Title: Kill Bill (volume 1) Release Year: 2003', 'Movie Title: Inception Release Year: 2010', and 'Movie Title: El Rahina Release Year: 2006'. The last two results are: 'Movie Title: Baby Driver Release Year: 2017' and 'Movie Title: Masgoon Tranzait Release Year: 2008'.

Genre:

Director:

Actor:



Search Results:

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

Movie Title: Masgoon Tranzait
Release Year: 2008

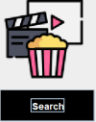
In the previous example, we selected movies of the action genre regardless of the actors and directors. The output displayed six movies that indeed belong to the action genre. Any movie that wasn't of the action genre didn't appear.

Movie Finder

Genre:

Director:

Actor:



☐ Action
☐ Comedy
☐ Drama
☐ Thriller
☐ Crime
☒ Christopher Nolan
☒ Edgar Wright
☐ Paul Thomas Anderson
☐ Quentin Tarantino
☐ Sandra Nashaat
☐ Taika Waititi
☐ Todd Phillips
☐ Edgar Wright
☐ John Travolta
☐ Leonardo DiCaprio
☐ Paul Thomas Anderson
☐ Quentin Tarantino
☐ Uma Thurman
☐ Taika Waititi

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:

Director:

Actor:



☐ Action
☐ Comedy
☐ Drama
☐ Thriller
☒ Crime
☒ Christopher Nolan
☒ Edgar Wright
☐ Paul Thomas Anderson
☐ Quentin Tarantino
☐ Sandra Nashaat
☐ Taika Waititi
☐ Todd Phillips
☐ Edgar Wright
☐ John Travolta
☐ Leonardo DiCaprio
☐ Paul Thomas Anderson
☐ Quentin Tarantino
☐ Uma Thurman
☐ Taika Waititi

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:

Director:

Actor:



Search

☐ 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 Waititi

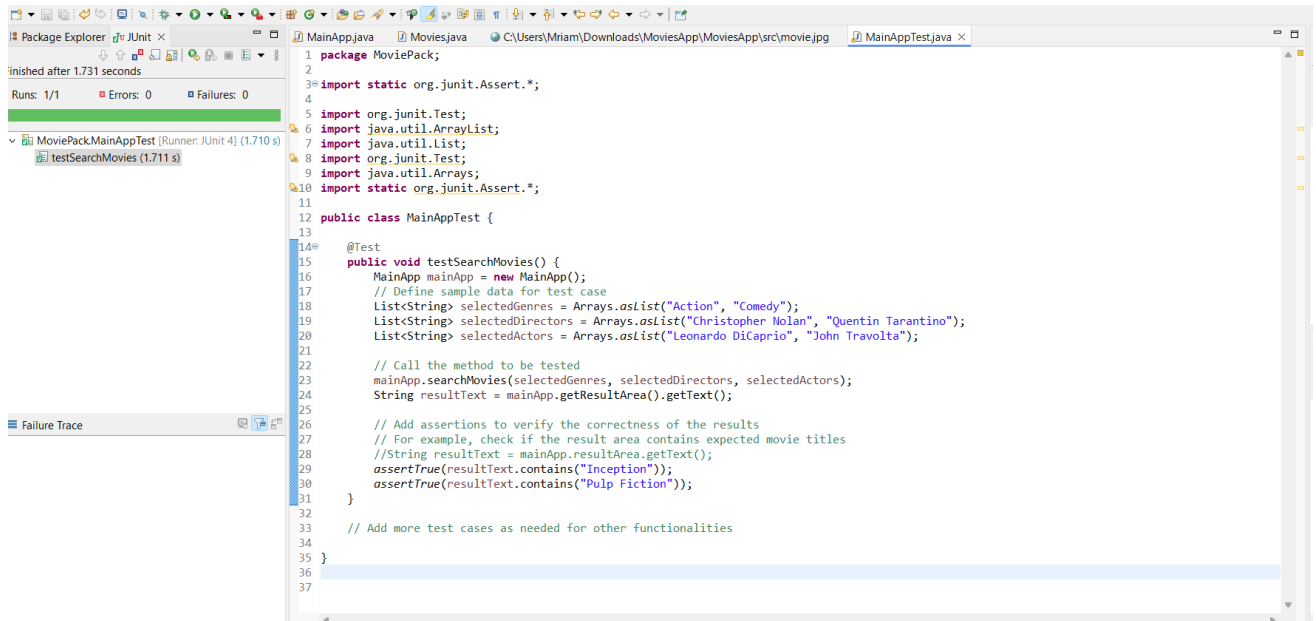
Search Results:

No movies found.

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

Testing : Test Cases

Test case 1



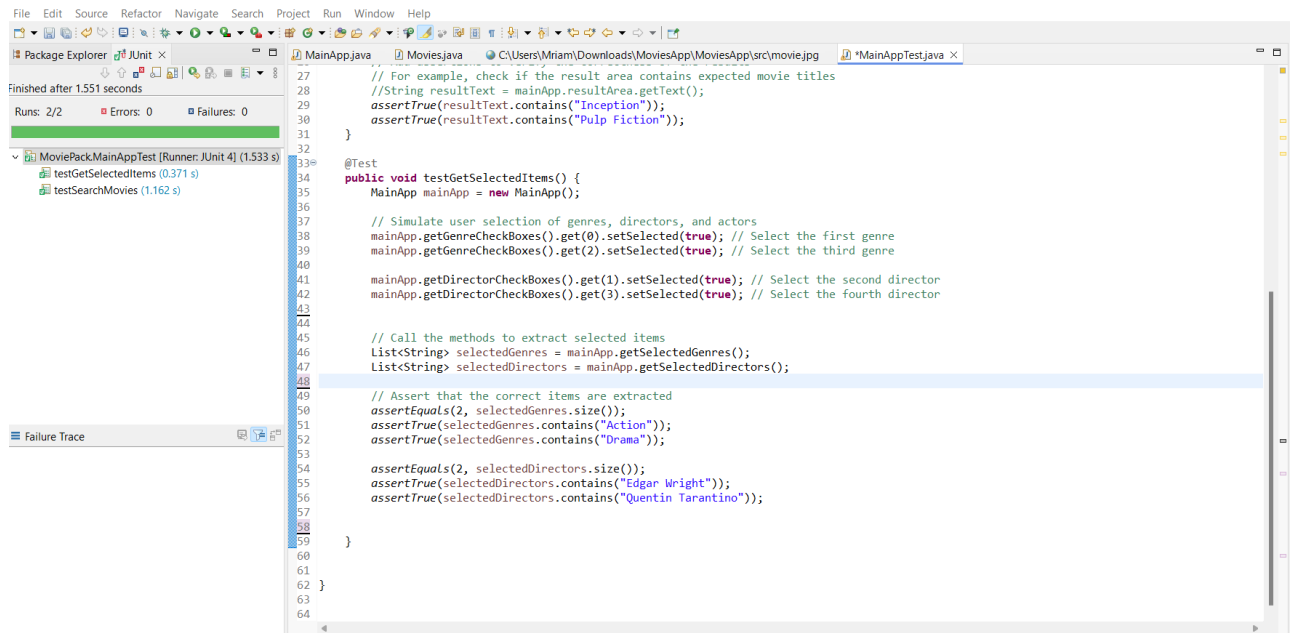
The screenshot shows an IDE with a JUnit test runner on the left and a Java code editor on the right. The test runner shows a successful run of the `testSearchMovies` test case. The code editor displays the `MainAppTest` class with a single test method `testSearchMovies` that uses `assertTrue` to verify the search results.

```
1 package MoviePack;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Test;
6 import java.util.ArrayList;
7 import java.util.List;
8 import org.junit.Test;
9 import java.util.Arrays;
10 import static org.junit.Assert.*;
11
12 public class MainAppTest {
13
14     @Test
15     public void testSearchMovies() {
16         MainApp mainApp = new MainApp();
17         // Define sample data for test case
18         List<String> selectedGenres = Arrays.asList("Action", "Comedy");
19         List<String> selectedDirectors = Arrays.asList("Christopher Nolan", "Quentin Tarantino");
20         List<String> selectedActors = Arrays.asList("Leonardo DiCaprio", "John Travolta");
21
22         // Call the method to be tested
23         mainApp.searchMovies(selectedGenres, selectedDirectors, selectedActors);
24         String resultText = mainApp.getResultArea().getText();
25
26         // Add assertions to verify the correctness of the results
27         // For example, check if the result area contains expected movie titles
28         //String resultText = mainApp.resultArea.getText();
29         assertTrue(resultText.contains("Inception"));
30         assertTrue(resultText.contains("Pulp Fiction"));
31     }
32
33     // Add more test cases as needed for other functionalities
34 }
35
36
37
```

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



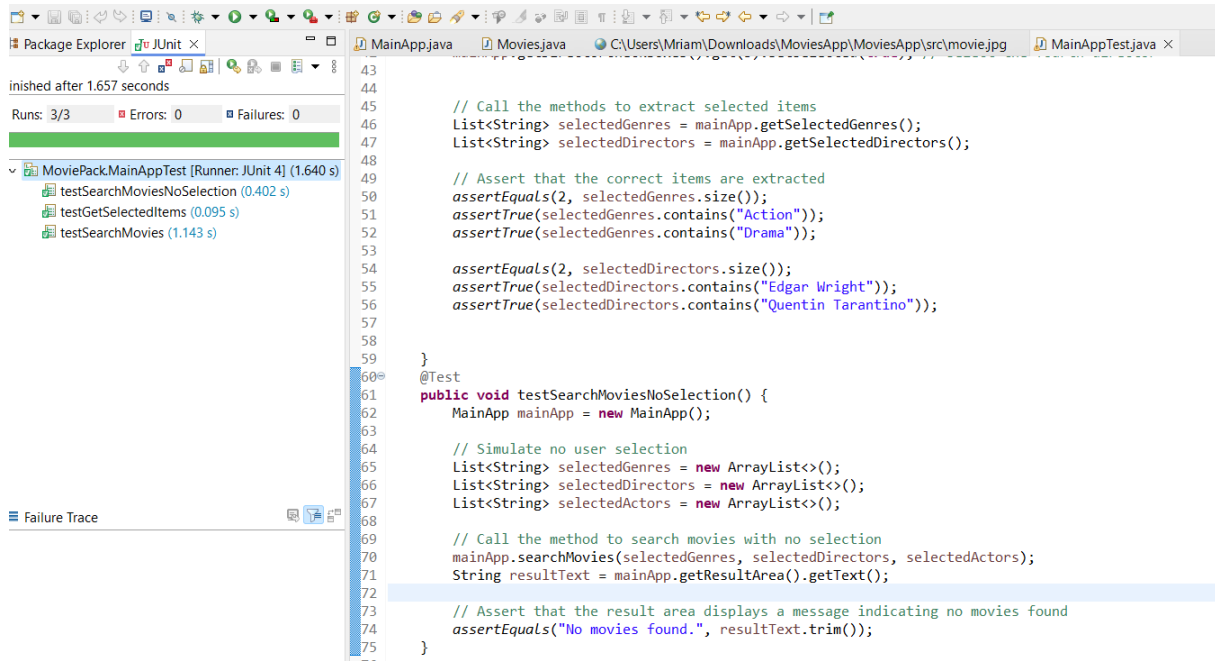
```
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JUnit x
Finished after 1.551 seconds
Runs: 2/2 Errors: 0 Failures: 0
MoviePack.MainAppTest [Runner: JUnit 4] (1.533 s)
  testGetSelectedItems (0.371 s)
  testSearchMovies (1.162 s)
Failure Trace
27
28 // For example, check if the result area contains expected movie titles
29 //String resultText = mainApp.resultArea.getText();
30 assertTrue(resultText.contains("Inception"));
31 assertTrue(resultText.contains("Pulp Fiction"));
32 }
33
34 @Test
35 public void testGetSelectedItems() {
36     MainApp mainApp = new MainApp();
37
38     // Simulate user selection of genres, directors, and actors
39     mainApp.getGenreCheckBoxes().get(0).setSelected(true); // Select the first genre
40     mainApp.getGenreCheckBoxes().get(2).setSelected(true); // Select the third genre
41
42     mainApp.getDirectorCheckBoxes().get(1).setSelected(true); // Select the second director
43     mainApp.getDirectorCheckBoxes().get(3).setSelected(true); // Select the fourth director
44
45     // Call the methods to extract selected items
46     List<String> selectedGenres = mainApp.getSelectedGenres();
47     List<String> selectedDirectors = mainApp.getSelectedDirectors();
48
49     // Assert that the correct items are extracted
50     assertEquals(2, selectedGenres.size());
51     assertTrue(selectedGenres.contains("Action"));
52     assertTrue(selectedGenres.contains("Drama"));
53
54     assertEquals(2, selectedDirectors.size());
55     assertTrue(selectedDirectors.contains("Edgar Wright"));
56     assertTrue(selectedDirectors.contains("Quentin Tarantino"));
57
58 }
59
60 }
61
62 }
63
64 }
```

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



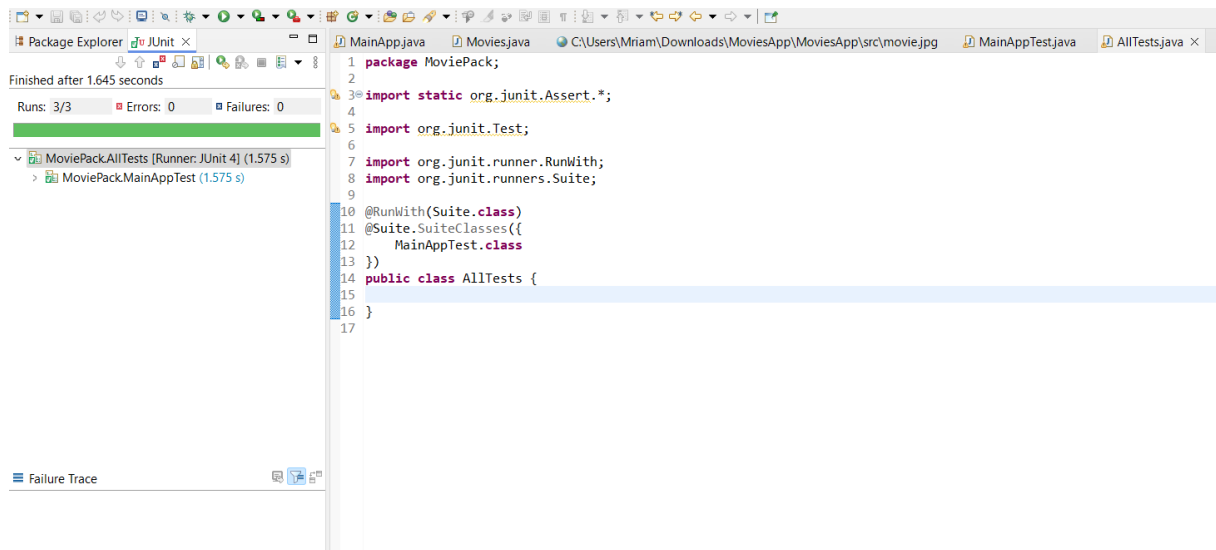
The screenshot shows an IDE with a JUnit test run on the left and the corresponding Java code on the right. The test run shows three tests passing: `testSearchMoviesNoSelection` (0.402 s), `testGetSelectedItems` (0.095 s), and `testSearchMovies` (1.143 s). The code on the right is `MainAppTest.java` and includes the following test case:

```
43
44
45 // Call the methods to extract selected items
46 List<String> selectedGenres = mainApp.getSelectedGenres();
47 List<String> selectedDirectors = mainApp.getSelectedDirectors();
48
49 // Assert that the correct items are extracted
50 assertEquals(2, selectedGenres.size());
51 assertTrue(selectedGenres.contains("Action"));
52 assertTrue(selectedGenres.contains("Drama"));
53
54 assertEquals(2, selectedDirectors.size());
55 assertTrue(selectedDirectors.contains("Edgar Wright"));
56 assertTrue(selectedDirectors.contains("Quentin Tarantino"));
57
58
59 }
60 @Test
61 public void testSearchMoviesNoSelection() {
62     MainApp mainApp = new MainApp();
63
64     // Simulate no user selection
65     List<String> selectedGenres = new ArrayList<>();
66     List<String> selectedDirectors = new ArrayList<>();
67     List<String> selectedActors = new ArrayList<>();
68
69     // Call the method to search movies with no selection
70     mainApp.searchMovies(selectedGenres, selectedDirectors, selectedActors);
71     String resultText = mainApp.getResultArea().getText();
72
73     // Assert that the result area displays a message indicating no movies found
74     assertEquals("No movies found.", resultText.trim());
75 }
```

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



```
1 package MoviePack;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Test;
6
7 import org.junit.runner.RunWith;
8 import org.junit.runners.Suite;
9
10 @RunWith(Suite.class)
11 @Suite.SuiteClasses({
12     MainAppTest.class
13 })
14 public class AllTests {
15
16 }
17
```

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.