



Ain Shams University

# CSE488: ONTOLOGIES AND THE SEMANTIC WEB

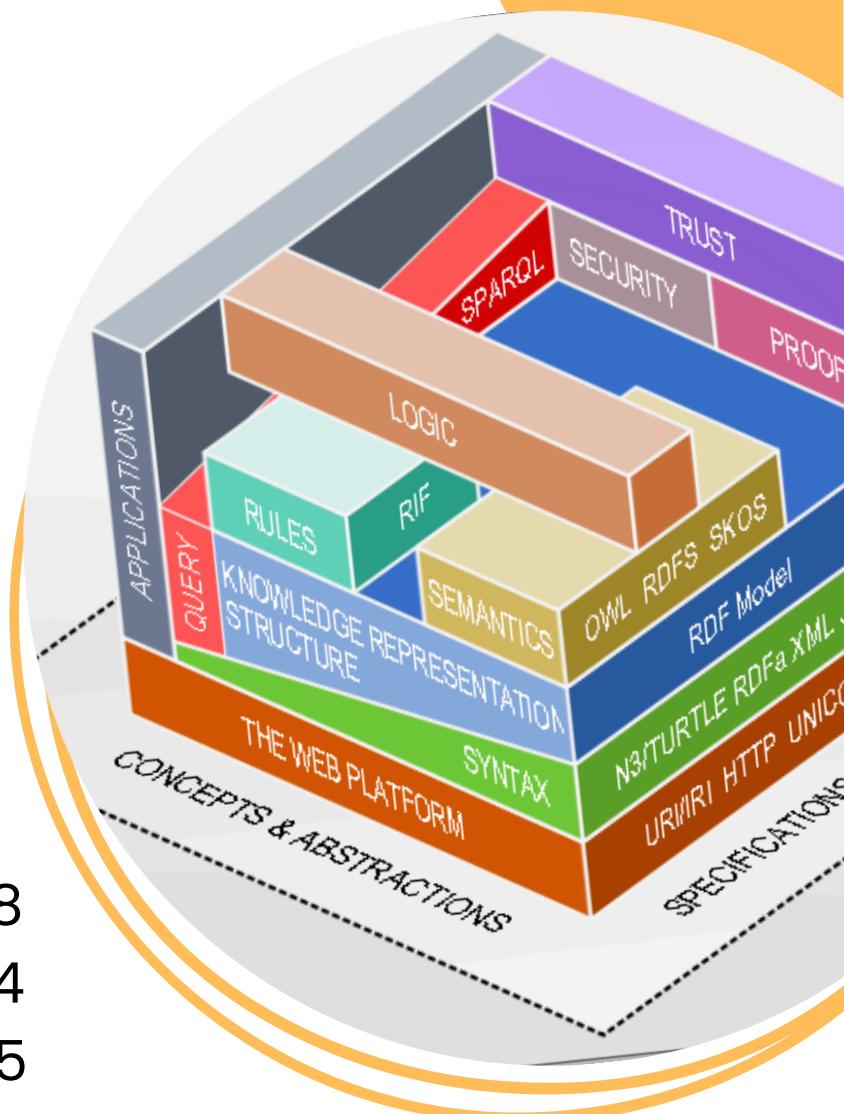
Presented to:

**Dr. Ensaf Hussein**

**Eng. Eman Khaled**

Prepared by:

Khaled Hassan	20p9978
Omar Ali Sobhy	20p2084
Habiba Yasser	20p9595
Laila Ihab	20p1005
Adham Yasser	20p9705



## Table of Contents

Problem Description: .....	2
Number of Entities: .....	2
Number of Relations: .....	3
Logic: .....	4
Axioms and Restrictions: .....	4
Check the consistency of the ontology: .....	8
Test Queries and Their Output: .....	9
Visualize the Ontology: .....	25
Tool used: .....	25
Graph: .....	26
Part IV: Manipulating the ontology using Jena .....	27
Test Cases: .....	48
Inference Correctness: .....	48
Ontology Consistency: .....	51
Query Performance: .....	51
Snapshots of the Interface: .....	52
1 <sup>st</sup> : Choose the actors to be included\excluded: .....	52
2 <sup>nd</sup> : Press Next and include/exclude directors: .....	56
3 <sup>rd</sup> : Press Next and include/exclude genres: .....	57
4 <sup>th</sup> : Press Next and Search to display the query result: .....	59
Bonus: What does the query string look like? .....	60
DFD (Data Flow Diagram): .....	61
The whole system: .....	61
Manage user component: .....	61
Explore ontology Component: .....	62
Additional Part: .....	63

### Github link:

[https://github.com/adham137/Ontology\\_application](https://github.com/adham137/Ontology_application)

# Semantic Web Project

## Problem Description:

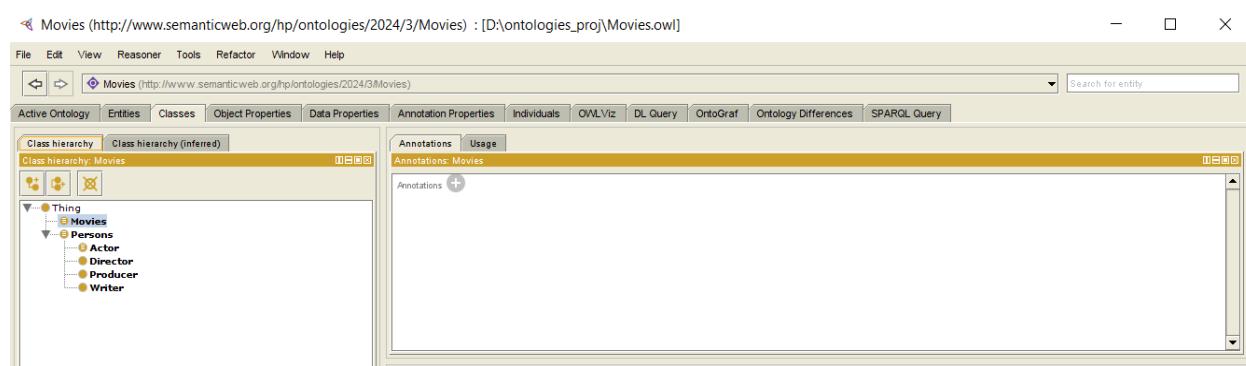
The problem domain that this ontology addresses revolves around the organization and representation of movie-related information. In the vast landscape of cinema, there exists a multitude of interconnected entities such as movies, directors, writers, actors, genres, and various attributes associated with them like titles, years, countries, and languages. Understanding and structuring this domain is crucial for various applications such as movie databases, recommendation systems, and analytical tools aimed at understanding trends in the film industry.

At its core, the problem entails creating a structured representation of movies and their associated entities to facilitate efficient querying, reasoning, and analysis. This involves defining the relationships between movies and their directors, writers, actors, genres, as well as capturing pertinent attributes of these entities such as names, genders, ages, and nationalities. Additionally, the ontology needs to encapsulate the taxonomy of movie genres to enable classification and categorization of movies based on their thematic content.

Finally, the problem domain encompasses considerations of data integrity, consistency, and interoperability. Ensuring that the ontology adheres to standard data modeling practices and is compatible with existing ontologies or data schemas within the domain of cinema is essential for promoting interoperability and facilitating seamless integration with other systems or datasets.

## Number of Entities:

- Thing class is the superclass of any classes made in the ontology.
- 2 classes are made:
  - a. Movies Class
  - b. Persons Class: Class that represents persons that could possibly be working in any movie, including actors, directors, writers and producers.
  - c. Movies and Persons are connected to each other via various object properties.
  - d. Movies and Persons are disjoint with each other, meaning that they don't have any common individual or instance.



Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: Movies

Description: Movies

- Baby\_Driver
- Boogie\_Nights
- El\_Mamar
- El\_Maslaha
- Forrest\_Gump
- Jojo\_Rabbit
- Kill\_Bill
- Pulp\_Fiction
- Shaun\_of\_the\_Dead
- There\_Will\_Be\_Blood
- Thor:Ragnarok

Target for key +

Disjoint With + Persons

To use the reasoner click Reasoner->Start reasoner  Show Inferences

## Number of Relations:

- The ontology consists of 8 Object Properties:
  - has Actor: which has **Domain**: Movies and **Range**: Persons
  - hasDirector: which has **Domain**: Movies and **Range**: Persons
  - hasWriter: which has **Domain**: Movies and **Range**: Persons
  - hasProducer: which has **Domain**: Movies and **Range**: Persons
  - isActorOf: which has **Domain**: Actor and **Range**: Movies
  - isDirectorOf: which has **Domain**: Director and **Range**: Movies
  - isWriterOf: which has **Domain**: Writer and **Range**: Movies
  - isProducerOf: which has **Domain**: Producer and **Range**: Movies

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: hasActor

Description: hasActor

Characteristics: hasActor

Equivalent To +

SubProperty Of +

Inverse Of + isActorOf

Domains (Intersection) + Persons

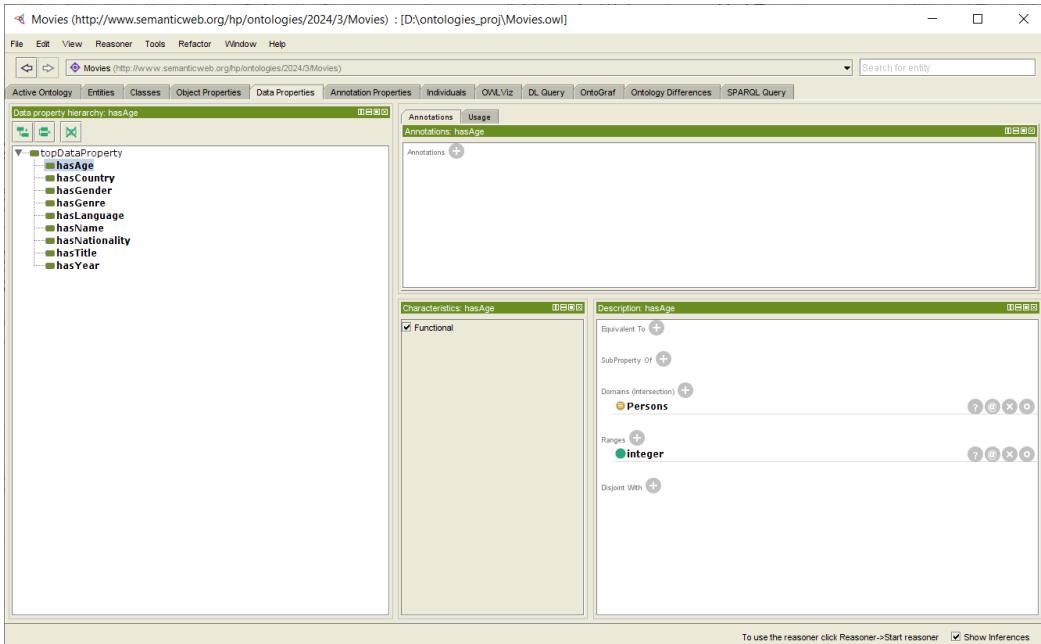
Ranges (Intersection) + Persons

Disjoint With +

SuperProperty Of (Chain) +

To use the reasoner click Reasoner->Start reasoner  Show Inferences

- The ontology also consists of 9 Data Properties:
  - hasAge: which has **Domain**: Persons and **Range**: xsd:integer
  - hasGender: which has **Domain**: Persons and **Range**: xsd:string
  - hasName: which has **Domain**: Persons and **Range**: xsd:string
  - hasNationality: which has **Domain**: Persons and **Range**: xsd:string
  - hasTitle: which has **Domain**: Movies and **Range**: xsd:string
  - hasGenre: which has **Domain**: Movies and **Range**: xsd:string
  - hasYear: which has **Domain**: Movies and **Range**: xsd:integer
  - hasCountry: which has **Domain**: Movies and **Range**: xsd:string
  - hasLanguage: which has **Domain**: Movies and **Range**: literal



## Logic:

To define the ontology for representing movies and associated entities, we can employ Description Logic (DL), a formalism commonly used for ontology modeling. DL allows us to define classes, properties, and axioms to capture the relationships and constraints within the domain.

## Axioms and Restrictions:

Define axioms to assert relationships and constraints between classes and properties. For example:

- Each Movie has one or several directors (hasDirector).
- Each Movie has one or several writers (hasWriter).
- Each Movie has one or several actors (hasActor).
- Each Movie has a title, a year, a country and a language.
- Each Movie belongs to one or several genres (hasGenre).
- To define the Genre of a movie, possible choices are: Thriller, Crime, Action, Drama or Comedy.

- Property `hasYear` is integer and from 0 to 2024

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query

Class hierarchy Movies

Annotations: Movies

Description: Movies

Equivalent To +

- hasGenre min 1 string
- hasActor min 1 Actor
- hasLanguage min 1 anyURI
- hasCountry min 1 string
- hasProducer min 1 Producer
- hasTitle exactly 1 string
- hasYear exactly 1 integer
- hasWriter min 1 Writer
- hasDirector min 1 Director

To use the reasoner click Reasoner->Start reasoner  Show Inferences

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query

Data property hierarchy: hasGenre

Annotations Usage

Description: hasGenre

Equivalent To +

SubProperty Of +

topDataProperty

Domains (intersection) +

- (hasGenre value "Action")
- or (hasGenre value "Comedy")
- or (hasGenre value "Crime")
- or (hasGenre value "Drama")
- or (hasGenre value "Horror")
- or (hasGenre value "Musical")
- or (hasGenre value "Romance")
- or (hasGenre value "Sci-Fi")
- or (hasGenre value "Thriller")

Movies

Ranges +

string

Disjoint With +

Reasoner state out of sync with active ontology  Show Inferences

The screenshot shows the Protégé interface for an ontology named 'Movies'. The left panel displays the 'Data property hierarchy' for 'hasYear', listing properties like hasAge, hasCountry, hasGender, hasGenre, hasLanguage, hasName, hasNationality, hasTitle, and hasYear. The right panel shows the 'Annotations' tab for 'hasYear', which is marked as functional. Annotations include:

- Domains (intersection):
  - hasYear some integer[> 0 , <= 2024]
  - Movies
- Ranges:
  - integer

A red box highlights the domain annotation.

- Actors, directors and writers are persons.
- Each Person has a gender (male or female), name, age, and nationality.
- Defined hasAge, hasName, hasGender, hasTitle and hasYear as **Functional** data properties.
- Defined Movies Class as Disjoint with Persons class.
- Defined the domain and Range for all properties as illustrated above.
- Defined has Actor, hasDirector, hasWriter, hasProducer as inverseOf: isActorOf, isDirectorOf, isWriterOf, isProducerOf.

The screenshot shows the Protégé interface for the same ontology. The left panel displays the 'Class hierarchy' with 'Persons' as a subclass of 'Thing'. The right panel shows the 'Description' tab for the 'Persons' class, which includes:

- Equivalent To:
  - hasNationality min 1 string
  - hasGender exactly 1 string
  - hasAge exactly 1 integer
  - hasName exactly 1 string

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_pro\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: hasActor

Characteristics: hasActor

- Functional
- Inverse functional
- Transitive
- Symmetric
- Asymmetric
- Reflexive
- Intransitive

Description: hasActor

- Equivalent To
- SubProperty Of
- Inverse Of
- isActorOf**
- Domains (Intersection)
- Movies**
- Ranges (Intersection)
- Persons**
- Disjoint With
- SuperProperty Of (None)

To use the reasoner click Reasoner->Start reasoner  Show Inferences

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_pro\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: hasGender

Characteristics: hasGender

- Functional

Description: hasGender

- Equivalent To
- SubProperty Of
- Domains (Intersection)
- Persons**
- (hasGender value "female") or (hasGender value "male")**
- Ranges
- string**

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_pro\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: hasAge

Characteristics: hasAge

- Functional

Description: hasAge

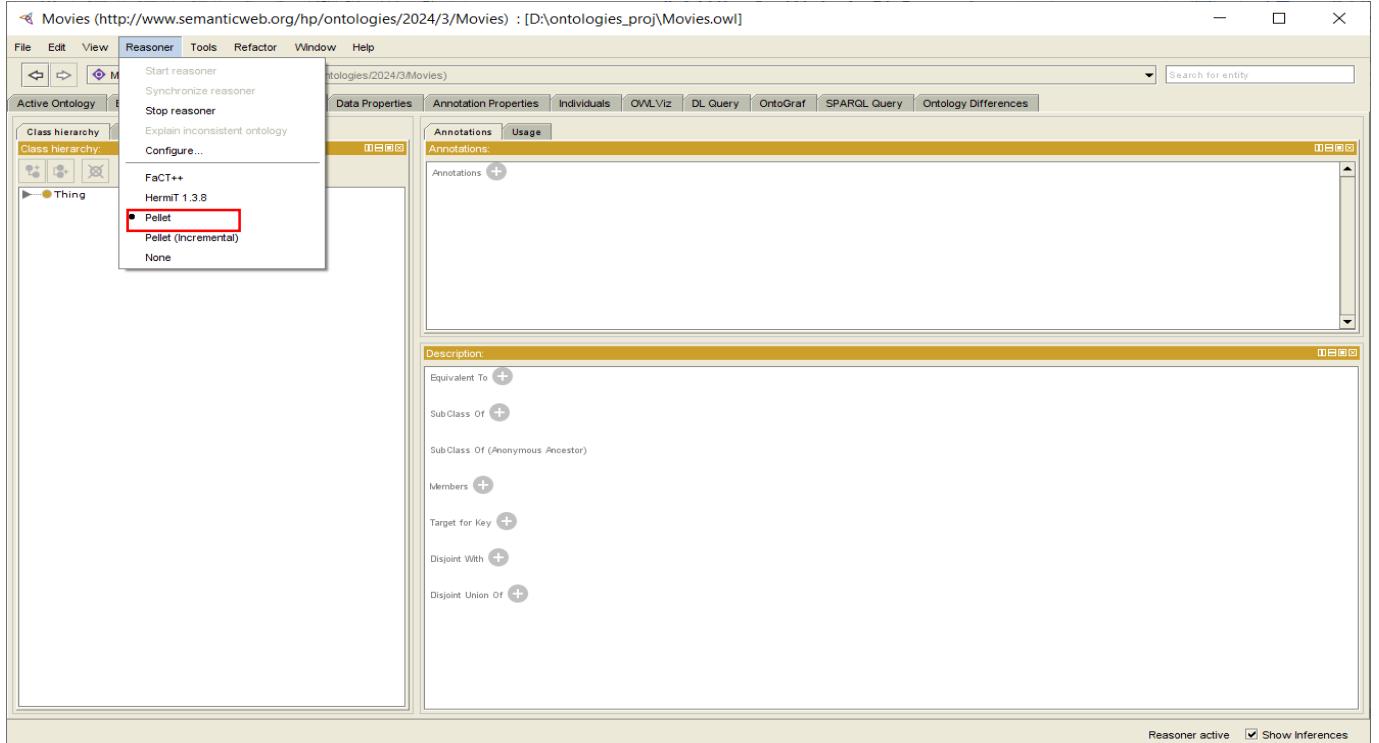
- Equivalent To
- SubProperty Of
- Domains (Intersection)
- Persons**
- Ranges
- integer**
- Disjoint With

To use the reasoner click Reasoner->Start reasoner  Show Inferences

Use existential and universal quantifiers to specify constraints on relationships and attributes. For example (shown above):

- Existential quantifiers can be used to specify that each Movie must have at least one director, writer, actor and genre.
- Universal quantifiers can be used to specify that each Person must have a gender, name, age, and nationality.

## Check the consistency of the ontology:



## Test Queries and Their Output:

- 1) Find Movies that are before year 2010 and have genre "Action".

```
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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?title ?year ?genre ?actor_name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasYear ?year.
?movie ont:hasTitle ?title.
?movie ont:hasGenre ?genre.
?movie ont:hasActor ?actor_name.
FILTER(?year < 2010 && ?genre= "Action")
}
```

The screenshot shows the OntoGraf interface with the SPARQL Query tab selected. The query window contains the SPARQL code provided above. The results pane displays a table with four columns: title, year, genre, and actor\_name. Two rows of data are shown:

title	year	genre	actor_name
"Africano"^^<http://www.w3.org/2001/XMLSchema#string>"2004"^^<http://www.w3.org/2001/XMLSchema#integer>	"Action"^^<http://www.w3.org/2001/XMLSchema#string>	Mona_Zaki	
"Kill Bill (volume1)"^^<http://www.w3.org/2001/XMLSchema#string>"2003"^^<http://www.w3.org/2001/XMLSchema#integer>	"Action"^^<http://www.w3.org/2001/XMLSchema#string>	Quentin_Tarantino	

2) 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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?actor_name
WHERE {
?movie rdf:type ont:Movies .
?movie ont:hasActor ?actor .
?actor ont:hasName ?actor_name .
}

```

The screenshot shows the OntoGraf interface with the following details:

- Title Bar:** Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]
- Menu Bar:** File, Edit, View, Reasoner, Tools, Refactor, Window, Help
- Toolbar:** Back, Forward, Home, Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>), Search for entity
- Tab Bar:** Active Ontology, Entities, Classes, Object Properties, Data Properties, Annotation Properties, Individuals, OWL Viz, DL Query, OntoGraf, SPARQL Query, Ontology Differences
- SPARQL Query Area:**

```

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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?actor_name
WHERE {
?movie rdf:type ont:Movies .
?movie ont:hasActor ?actor .
?actor ont:hasName ?actor_name .
}

```
- Results Table:**

actor_name
"Mona Zaki"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"John Travolta"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Scarlett Johansson"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Taika Waititi"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Quentin Tarantino"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Quentin Tarantino"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Tom Hanks"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Ahmed Ezz"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Uma Thurman"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Ahmed Ezz"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >

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 ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT DISTINCT ?director_name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasDirector ?director.
?director ont:hasName ?director_name.
}
```

The screenshot shows the OntoGraf interface with the following details:

- Toolbar:** File, Edit, View, Reasoner, Tools, Refactor, Window, Help.
- Title Bar:** Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_pro\Movies.owl]
- Search Bar:** Search for entity
- Tab Bar:** Active Ontology, Entities, Classes, Object Properties, Data Properties, Annotation Properties, Individuals, OWLViz, DL Query, OntoGraf, SPARQL Query, Ontology Differences.
- SPARQL query area:** SPARQL query (highlighted in blue).

```
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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT DISTINCT ?director_name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasDirector ?director.
?director ont:hasName ?director_name.
}
```
- Results Table:** A table titled "director\_name" showing the names of directors. The table contains the following data:

director_name
"Taika Waititi"
"Quentin Tarantino"
"Edgar Wright"
"Paul Thomas Anderson"

4) 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 ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT DISTINCT ?writer_name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasWriter ?writer.
?writer ont:hasName ?writer_name.
}
```

The screenshot shows the OntoGraf application window. The title bar reads "Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]". The menu bar includes File, Edit, View, Reasoner, Tools, Refactor, Window, Help. The toolbar has icons for back, forward, and search. The main area has tabs for Active Ontology, Entities, Classes, Object Properties, Data Properties, Annotation Properties, Individuals, OWL/Viz, DL Query, OntoGraf, SPARQL Query, and Ontology Differences. The "SPARQL query" tab is selected. The query text is identical to the one above. Below the query, the results are displayed in a table with a single column labeled "writer\_name". The data rows are: "Quentin Tarantino"^^<[http://www.w3.org/2001/XMLSchema#stringhttp://www.w3.org/2001/XMLSchema#string](http://www.w3.org/2001/XMLSchema#string)>, "Paul Thomas Anderson"^^<<http://www.w3.org/2001/XMLSchema#string>>, and "Taika Waititi"^^<<http://www.w3.org/2001/XMLSchema#string>>.

5) 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 ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?movie ?director_name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasDirector ?director.
?director ont:hasName ?director_name.
?movie ont:hasGenre ?genre.
FILTER(?genre = "Thriller")
}

```

The screenshot shows the Protégé interface with the following details:

- Title Bar:** Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_pro\Movies.owl]
- Menu Bar:** File Edit View Reasoner Tools Refactor Window Help
- Toolbar:** Buttons for back, forward, search, and other navigation.
- Tab Bar:** Active Ontology, Entities, Classes, Object Properties, Data Properties, Annotation Properties, Individuals, OWLViz, DL Query, OntoGraf, SPARQL Query, Ontology Differences.
- SPARQL query tab:** Contains the SPARQL query shown in the code block above.
- Results Table:**

movie	director_name
Kill_Bill	"Quentin Tarantino"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Pulp_Fiction	"Quentin Tarantino"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Baby_Driver	"Edgar Wright"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
There_Will_Be_Blood	"Paul Thomas Anderson"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >

6) 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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?title
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasTitle ?title.
?movie ont:hasGenre ?genre1, ?genre2.
FILTER(?genre1 = "Crime" && ?genre2 = "Thriller")
}
```

The screenshot shows the Protégé ontology editor interface. The top menu bar includes File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. A tab bar at the top shows the active ontology is 'Movies' (http://www.semanticweb.org/hp/ontologies/2024/3/Movies). Below the tabs is a search bar labeled 'Search for entity'. The main workspace has several tabs: Active Ontology, Entities, Classes, Object Properties, Data Properties, Annotation Properties, Individuals, OMLViz, DL Query, OntoGraf, SPARQL Query, and Ontology Differences. The 'SPARQL query' tab is selected, displaying the SPARQL code from the previous block. The results pane below shows the output of the query:

title
"Pulp Fiction"^^<http://www.w3.org/2001/XMLSchema#string>
"Kill Bill (volume1)"^^<http://www.w3.org/2001/XMLSchema#string>

7) 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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?actor_name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasTitle ?title.
?movie ont:hasActor ?actor.
?actor ont:hasGender ?gender.
?actor ont:hasName ?actor_name.
FILTER(?title = "Jojo Rabbit" && ?gender="male")
}

```

The screenshot shows the OntoGraf interface with the 'Movies' ontology loaded. The main window has several tabs at the top: Active Ontology, Entities, Classes, Object Properties, Data Properties, Annotation Properties, Individuals, OWL/Viz, DL Query, OntoGraf, SPARQL Query, and Ontology Differences. The 'Individuals' tab is active, displaying a list of individuals including 'Africano', 'Ahmed\_Ezz', 'Ansel\_Elgort', 'Baby\_Driver', 'Boogie\_Nights', 'Edgar\_Wright', 'El\_Mamar', 'El\_Maslah', 'Forrest\_Gump', 'John\_Travolta', 'Jojo\_Rabbit' (which is selected and highlighted), 'Kill\_Bill', 'Mona\_Zaki', 'Paul\_Thomas\_Anderson', 'Pulp\_Fiction', 'Quentin\_Tarantino', 'Scarlett\_Johansson', 'Shaun\_of\_the\_Dead', 'Taika\_Waititi', 'There\_Will\_be\_Blood', 'Thor\_Ragnarok', 'Tom\_Hanks', and 'Uma\_Thurman'. Below this list, there are sections for 'Types' (Movies), 'Same Individual As', and 'Different Individuals'. The 'Annotations' tab shows annotations for 'Jojo\_Rabbit'. The 'Description' tab shows 'Jojo\_Rabbit' is a 'Movies' type. The 'Property assertions' tab shows object property assertions for 'Jojo\_Rabbit' involving 'Taika\_Waititi' and 'Scarlett\_Johansson'. The 'Data property assertions' tab shows data properties like 'hasYear 2019', 'hasLanguage "English"@en', 'hasCountry "USA"^^string', 'hasTitle "Jojo Rabbit"^^string', and 'hasGenre "Comedy"^^string'. The bottom status bar indicates 'Reasoner state out of sync with active ontology' and 'Show Inferences'.

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) Search for entity

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf SPARQL Query Ontology Differences

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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?actor_name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:title ?title.
?movie ont:hasActor ?actor.
?actor ont:hasGender ?gender.
?actor ont:hasName ?actor_name.
FILTER(?title = "Jojo Rabbit" & ?gender = "male")
}

```

"Taika Waititi"^^<<http://www.w3.org/2001/XMLSchema#string>>

### 8) How many movies have both "Action" and "Thriller" as genres?

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) Search for entity

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf SPARQL Query Ontology Differences

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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT (COUNT(DISTINCT ?movie) AS ?count)
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasGenre ?genre1, ?genre2.
FILTER(?genre1 = "Action" & ?genre2 = "Thriller")
}

```

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

### 9) 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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?movieTitle
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasWriter ?writer.
?writer ont:hasName ?writer_name.
?movie ont:hasTitle ?movieTitle.
FILTER(?writer_name = "Quentin Tarantino")
}

```

The screenshot shows the OntoGraf interface with the following details:

- Title Bar:** Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]
- Menu Bar:** File, Edit, View, Reasoner, Tools, Refactor, Window, Help
- Toolbar:** Back, Forward, Home, Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>), Search for entity
- Tab Bar:** Active Ontology, Entities, Classes, Object Properties, Data Properties, Annotation Properties, Individuals, OWLviz, DL Query, OntoGraf, SPARQL Query, Ontology Differences
- SPARQL query area:**

```
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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?movieTitle
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasWriter ?writer.
?writer ont:hasName ?writer_name.
?movie ont:hasTitle ?movieTitle.
FILTER(?writer_name = "Quentin Tarantino")
}
```
- Result pane:** movieTitle
  - "Kill Bill (volume1)"^^<<http://www.w3.org/2001/XMLSchema#string>>
  - "Pulp Fiction"^^<<http://www.w3.org/2001/XMLSchema#string>>

## 10) 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 ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies
#>

SELECT ?movieTitle

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasLanguage ?language.

?movie ont:hasTitle ?movieTitle.

FILTER(LANG(?language) = "ar")

}
```

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) Search for entity

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLviz DL Query OntoGraf SPARQL Query Ontology Differences

**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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?movieTitle
WHERE {
?movie rdf:type ont:Movie.
?movie onthasLanguage ?language.
?movie onthasTitle ?movieTitle.
FILTER(LANG(?language) = "ar")
}
```

movieTitle

```
"Africano"^^<http://www.w3.org/2001/XMLSchema#string>
"El Maslaha"^^<http://www.w3.org/2001/XMLSchema#string>
"El Mamar"^^<http://www.w3.org/2001/XMLSchema#string>
```

## 11) List the name of Actors older than 51 years.

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) Search for entity

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLviz DL Query OntoGraf SPARQL Query Ontology Differences

**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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?actorName
WHERE {
?actor rdf:type ont:Actor.
?actor onthasAge ?age.
?actor onthasName ?actorName.
FILTER(?age > 51)
}
```

actorName

```
"Quentin Tarantino"^^<http://www.w3.org/2001/XMLSchema#string>
"John Travolta"^^<http://www.w3.org/2001/XMLSchema#string>
"Tom Hanks"^^<http://www.w3.org/2001/XMLSchema#string>
"Ahmed Ezz"^^<http://www.w3.org/2001/XMLSchema#string>
```

12) A query that contains at least 2 Optional Graph Patterns

- a) 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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT DISTINCT ?name ?age ?nationality
WHERE {
    ?actor rdf:type ont:Actor.
    ?actor ont:hasName ?name.
    OPTIONAL {
        ?actor ont:hasAge ?age
    }
    OPTIONAL {
        ?actor ont:hasNationality ?nationality
    }
}

```

The screenshot shows the Protégé SPARQL Query interface with the following details:

- Title Bar:** Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]
- Toolbar:** File, Edit, View, Reasoner, Tools, Refactor, Window, Help
- Address Bar:** Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>)
- Search Bar:** Search for entity
- Tab Bar:** Active Ontology, Entities, Classes, Object Properties, Data Properties, Annotation Properties, Individuals, OWL Viz, DL Query, OntoGraff, SPARQL Query, Ontology Differences
- SPARQL Query Tab:** SPARQL query
- 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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT DISTINCT ?name ?age ?nationality
WHERE {
    ?actor rdf:type ont:Actor.
    ?actor ont:hasName ?name.
    OPTIONAL {
        ?actor ont:hasAge ?age
    }
    OPTIONAL {
        ?actor ont:hasNationality ?nationality
    }
}

```
- Results Table:**

name	age	nationality
"Uma Thurman"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"43"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"American"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Scarlett Johansson"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"39"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"American"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Mona Zaki"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"47"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"Egyptian"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Tom Hanks"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"67"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"American"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Talika Waititi"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"47"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"New Zealand"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"John Travolta"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"70"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"American"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Edgar Wright"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"48"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"British"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Quentin Tarantino"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"53"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"American"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Paul Thomas Anderson"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"51"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"American"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
"Ahmed Ezz"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >	"52"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >	"Egyptian"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >

- b) A query that selects movies and includes optional graph to retrieve the director's name and writer for each movie, if available.

```

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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT DISTINCT ?movieTitle ?directorName ?writerName
WHERE{
    ?movie rdf:type ont:Movies.
    ?movie ont:hasTitle ?movieTitle.
    OPTIONAL {
        ?movie ont:hasDirector ?director.
        ?director ont:hasName ?directorName.
    }
    OPTIONAL {
        ?movie ont:hasWriter ?writer.
        ?writer ont:hasName ?writerName.
    }
}

```

The screenshot shows the Protégé SPARQL Query interface with the following details:

- Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies\_pro\Movies.owl]**
- 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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT DISTINCT ?movieTitle ?directorName ?writerName
WHERE{
    ?movie rdf:type ont:Movies.
    ?movie ont:hasTitle ?movieTitle.
    OPTIONAL {
        ?movie ont:hasDirector ?director.
        ?director ont:hasName ?directorName.
    }
    OPTIONAL {
        ?movie ont:hasWriter ?writer.
        ?writer ont:hasName ?writerName.
    }
}

```
- Results Table:**

movieTitle	directorName	writerName
"El Mamar"		
"El Maslaha"		
"Forrest Gump"		
"Pulp Fiction"	"Quentin Tarantino"	
"Thor: Ragnarok"	"Taika Waititi"	
"Africano"		
"Boogie Nights"	"Paul Thomas Anderson"	
"Baby Driver"	"Edgar Wright"	
"Shaun of the Dead"	"Edgar Wright"	
"Kill Bill (volume1)"	"Quentin Tarantino"	"Quentin Tarantino"
"Kill Bill (volume1)"	"Uma Thurman"	"Uma Thurman"
"There Will Be Blood"	"Paul Thomas Anderson"	"Paul Thomas Anderson"
"Jojo Rabbit"	"Taika Waititi"	"Taika Waititi"
- Buttons:** Execute, Reasoner active, Show Inferences

13) A query that contains at least 2 alternatives(UNION) and conjunctions(AND). The query selects movies and retrieves their titles, along with either the associated actors or the directors The query also includes a conjunction by using the FILTER function to make sure movie was before 2010 and has genre "Action"

```
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 ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?title ?year ?genre ?actor_name ?director_name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasTitle ?title.
?movie ont:hasYear ?year.
?movie ont:hasGenre ?genre.
{
?movie ont:hasActor ?actor_name.
}
UNION
{
?movie ont:hasDirector ?director_name.
}
FILTER(?year < 2010 && ?genre = "Action")
}
```

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_pro\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) Search for entity

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLviz DL Query OntoGraf SPARQL Query Ontology Differences

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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?title ?year ?genre ?actor_name ?director_name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasTitle ?title.
?movie ont:hasYear ?year.
?movie ont:hasGenre ?genre.
?
?movie ont:hasActor ?actor_name.
}
UNION
?
?movie ont:hasDirector ?director_name.
}
FILTER(?year < 2010 && ?genre = "Action")
}
```

title	year	genre	actor_name	director_name
"Africano"^^< <a href="http://www.w3.org/2001/XMLSchema#2004">http://www.w3.org/2001/XMLSchema#2004</a> ^^< <a href="http://www.w3.org/2001/XMLSchema#Action">http://www.w3.org/2001/XMLSchema#Action</a> ^^< <a href="http://www.w3.org/2001/XMLSchema#stringMona_Zaki">http://www.w3.org/2001/XMLSchema#stringMona_Zaki</a>				
"Kill Bill (volume1)"^^< <a href="http://www.w3.org/2001/XMLSchema#2003">http://www.w3.org/2001/XMLSchema#2003</a> ^^< <a href="http://www.w3.org/2001/XMLSchema#Action">http://www.w3.org/2001/XMLSchema#Action</a> ^^< <a href="http://www.w3.org/2001/XMLSchema#stringQuentin_Tarantino">http://www.w3.org/2001/XMLSchema#stringQuentin_Tarantino</a>				
"Kill Bill (volume2)"^^< <a href="http://www.w3.org/2001/XMLSchema#2003">http://www.w3.org/2001/XMLSchema#2003</a> ^^< <a href="http://www.w3.org/2001/XMLSchema#Action">http://www.w3.org/2001/XMLSchema#Action</a> ^^< <a href="http://www.w3.org/2001/XMLSchema#stringQuentin_Tarantino">http://www.w3.org/2001/XMLSchema#stringQuentin_Tarantino</a>				

Execute

Reasoner active  Show Inferences

14) A query that contains a CONSTRUCT query form, construct triples for each movie, including its type, title, genre, and 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 ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
CONSTRUCT {
    ?movie rdf:type ont:Movies.
    ?movie ont:hasTitle ?title.
    ?movie ont:hasGenre ?genre.
    ?movie ont:hasActor ?actor.
}
WHERE {
    ?movie rdf:type ont:Movies.
    ?movie ont:hasTitle ?title.
    ?movie ont:hasGenre ?genre.
    ?movie ont:hasActor ?actor.
}
```

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf SPARQL Query Ontology Differences

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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
CONSTRUCT {
?movie rdf:type ont:Movies.
?movie ont:hasTitle ?title.
?movie ont:hasGenre ?genre.
?movie ont:hasActor ?actor.
}
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasTitle ?title.
?movie ont:hasGenre ?genre.
?movie ont:hasActor ?actor.
}
```

Subject	Predicate	Object
Africano	type	Movies
Africano	hasTitle	"Africano"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Africano	hasGenre	"Action"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Africano	hasActor	Mona_Zaki
Pulp_Fiction	type	Movies
Pulp_Fiction	hasTitle	"Pulp Fiction"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Pulp_Fiction	hasGenre	"Crime"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Pulp_Fiction	hasActor	John_Travolta
Pulp_Fiction	hasGenre	"Thriller"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Jojo_Rabbit	type	Movies
Jojo_Rabbit	hasTitle	"Jojo Rabbit"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Jojo_Rabbit	hasGenre	"Comedy"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Jojo_Rabbit	hasActor	Scarlett_Johansson
Jojo_Rabbit	hasActor	Taika_Waititi
Kill_Bill	type	Movies
Kill_Bill	hasTitle	"Kill Bill Vol. 1"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Kill_Bill	hasGenre	"Action"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Kill_Bill	hasActor	Uma_Thurman

Execute

Reasoner active  Show Inferences

- 15) A query that contains an ASK query form: The ASK query form in SPARQL is used to ask a true/false question about the existence of a certain pattern in the data. The ASK query form will return either true or false, indicating whether there are movies with the genre "Thriller" in the data.

```
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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

ASK

WHERE {
?movie rdf:type ont:Movies.

?movie ont:hasTitle ?title.

?movie ont:hasGenre ?genre.

FILTER(?genre = "Thriller")

}
```

The screenshot shows the Protégé interface with the following details:

- Title Bar:** Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]
- Menu Bar:** File Edit View Reasoner Tools Refactor Window Help
- Toolbar:** Back Forward Home Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) Search for entity
- Tab Bar:** Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWL/Viz DL Query OntoGraf SPARQL Query Ontology Differences
- SPARQL query area:**

```

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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
ASK
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasTitle ?title.
?movie ont:hasGenre ?genre.
FILTER(?genre = 'Thriller')
}

```
- Result Table:**

Result	
True	

- 16) A query that contains a DESCRIBE query form to retrieve RDF data that describes a specific resource (?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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

DESCRIBE ?actor

WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasActor ?actor.
?actor ont:hasName ?name.
?actor ont:hasAge ?age.
FILTER(?age > 50)
}

```

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies.owl]

File Edit View Reasoner Tools Refactor Window Help

Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLviz DL Query OntoGraf SPARQL Query Ontology Differences

Search for entity

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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
DESCRIBE ?actor
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasActor ?actor.
?actor ont:hasName ?name.
?actor ont:hasAge ?age.
FILTER(?age > 50)
}
```

Subject	Predicate	Object
John_Travolta	type	NamedIndividual
Tom_Hanks	type	NamedIndividual
Ahmed_Ezz	hasGender	"male"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Ahmed_Ezz	type	NamedIndividual
Quentin_Tarantino	type	NamedIndividual
John_Travolta	type	NamedIndividual
Quentin_Tarantino	type	NamedIndividual
Quentin_Tarantino	hasAge	"53"^^< <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> >
Quentin_Tarantino	type	NamedIndividual
Tom_Hanks	hasGender	"male"^^< <a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a> >
Tom_Hanks	type	NamedIndividual
Tom_Hanks	type	Director
Tom_Hanks	type	NamedIndividual
Quentin_Tarantino	type	Actor
Quentin_Tarantino	type	NamedIndividual

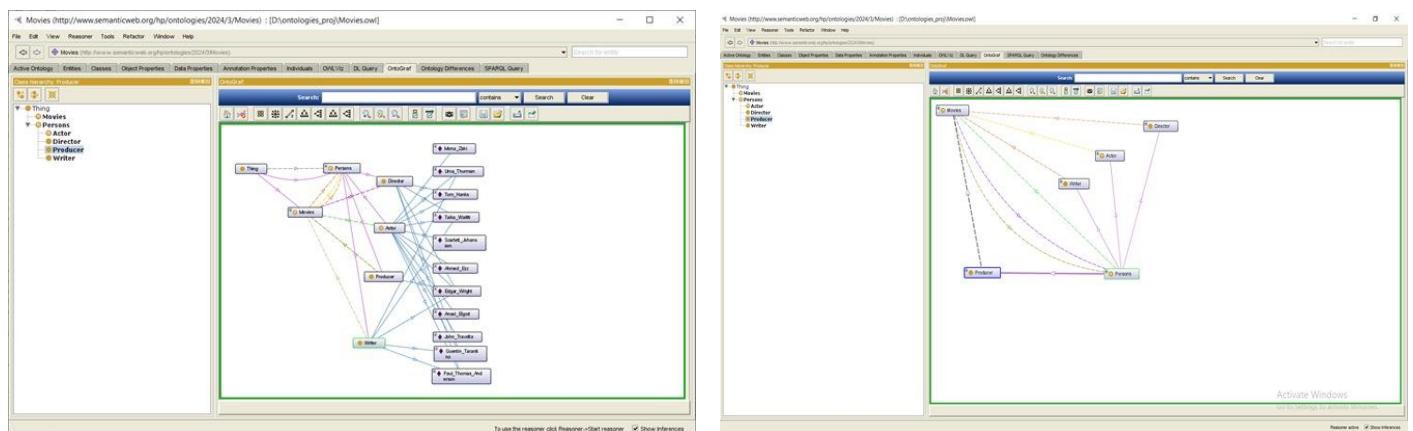
Execute

Reasoner active  Show Inferences

## Visualize the Ontology:

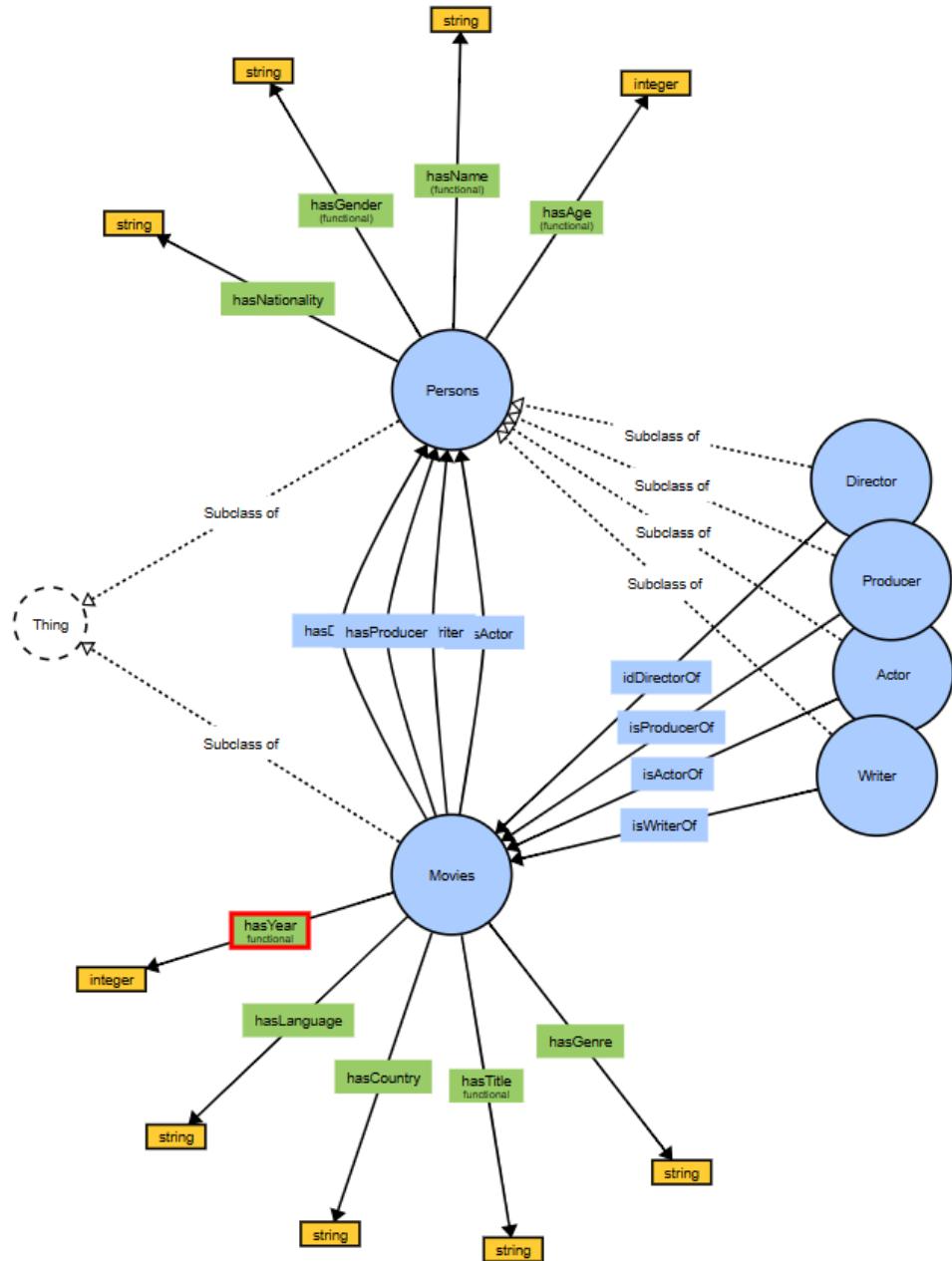
Tool used:

<https://service.tib.eu/webowl/>



## Graph:

[https://service.tib.eu/webowl/#opts=editorMode=true;#new\\_ontology1](https://service.tib.eu/webowl/#opts=editorMode=true;#new_ontology1)



## Part IV: Manipulating the ontology using Jena

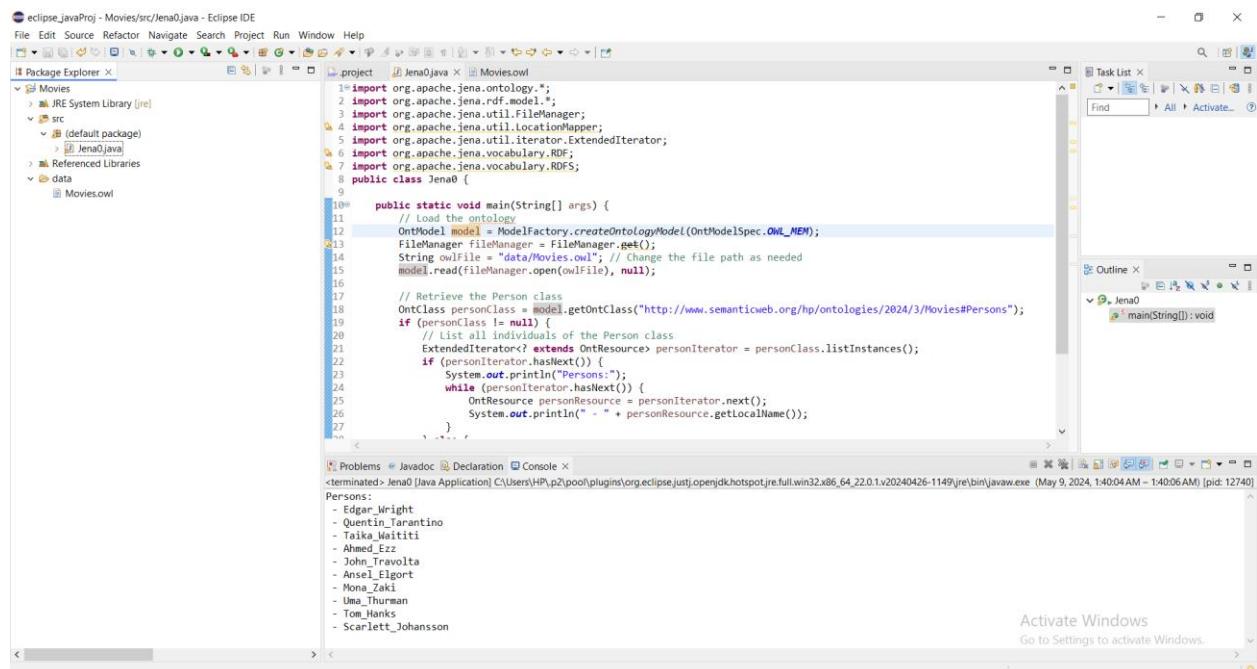
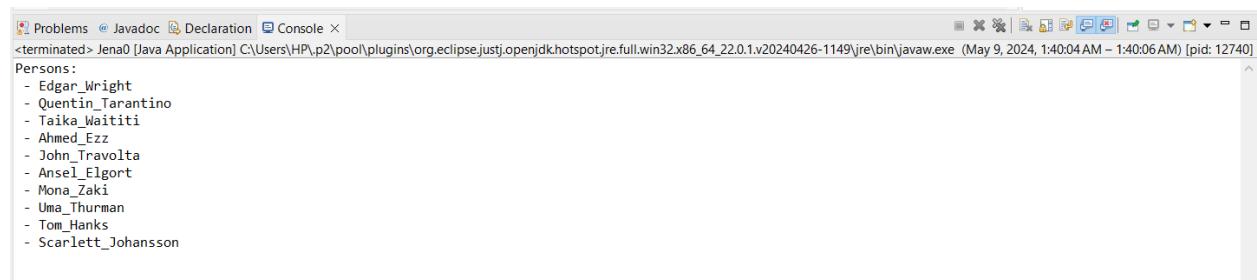
1. Create a java program (Jena1.java) that loads the ontology and displays all the Persons (without using queries, without inference).

```
import org.apache.jena.ontology.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.util.FileManager;
import org.apache.jena.util.LocationMapper;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
public class Jena0 {

    public static void main(String[] args) {
        // Load the ontology
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        FileManager fileManager = FileManager.get();
        String owlFile = "data/Movies.owl"; // Change the file path as needed
        model.read(fileManager.open(owlFile), null);

        // Retrieve the Person class
        OntClass personClass =
model.getOntClass("http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Persons");
        if (personClass != null) {
            // List all individuals of the Person class
            ExtendedIterator<? extends OntResource> personIterator = personClass.listInstances();
            if (personIterator.hasNext()) {
                System.out.println("Persons:");
            }
        }
    }
}
```

```
while (personIterator.hasNext()) {  
    OntResource personResource = personIterator.next();  
    System.out.println(" - " + personResource.getLocalName());  
}  
}  
}  
}  
}  
}  
}
```



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.

```
import org.apache.jena.ontology.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.query.*;
import org.apache.jena.util.FileManager;

import java.io.InputStream;

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

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/Movies.owl";
        model.read(fileManager.open(owlFile), null);

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

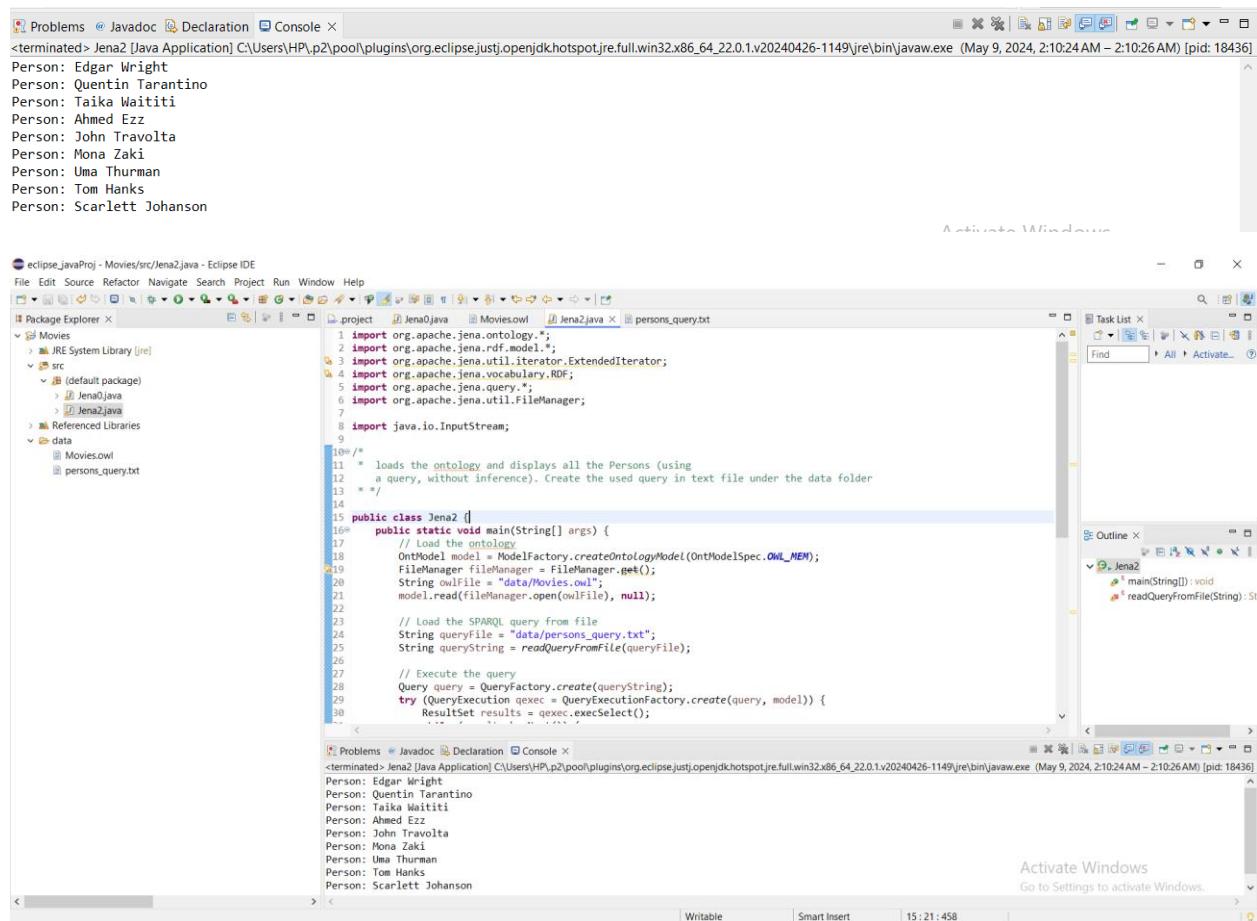
        // Execute the query
        Query query = QueryFactory.create(queryString);
        try (QueryExecution qexec = QueryExecutionFactory.create(query, model)) {
            ResultSet results = qexec.execSelect();
            while (results.hasNext()) {
                QuerySolution solution = results.next();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

    // Helper method to read the query from file
    private static String readQueryFromFile(String queryFile) {
        StringBuilder sb = new StringBuilder();
```

```

try (InputStream inputStream = FileManager.get().open(queryFile)) {
    if (inputStream != null) {
        String line;
        java.io.BufferedReader reader = new java.io.BufferedReader(
            new java.io.InputStreamReader(inputStream, "UTF-8"));
        while ((line = reader.readLine()) != null) {
            sb.append(line).append("\n");
        }
    }
} catch (java.io.IOException e) {
    e.printStackTrace();
}
return sb.toString();
}
}

```



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.readInferredModelFromRuleFile method and use owl rules

```
import org.apache.jena.ontology.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.rulesys.GenericRuleReasonerFactory;
import org.apache.jena.util.FileManager;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.jena.vocabulary.RDF;

/*
 * loads the ontology and displays all the Actors
 * (without using queries, using inference).
 */

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/Movies.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/hp/ontologies/2024/3/Movies#Actor");

    ExtendedIterator<? extends OntResource> actorIterator = actorClass.listInstances();

    while (actorIterator.hasNext()) {

        OntResource actorResource = actorIterator.next();

        Property nameProperty =
model.getProperty("http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasName");

        StmtIterator actorNameIterator = actorResource.listProperties(nameProperty);

        while (actorNameIterator.hasNext()) {

            Statement actorNameStatement = actorNameIterator.next();

            RDFNode actorNameNode = actorNameStatement.getObject();

            if (actorNameNode != null && actorNameNode.isLiteral()) {

                String actorName = actorNameNode.asLiteral().getString();

                System.out.println("Actor: " + actorName);

            }

        }

    }

}

```

Actors listed in the terminal:

- Actor: Paul Thomas Anderson
- Actor: Scarlett Johanson
- Actor: Mona Zaki
- Actor: John Travolta
- Actor: Tom Hanks
- Actor: Ahmed Ezz
- Actor: Uma Thurman
- Actor: Taika Waititi
- Actor: Quentin Tarantino
- Actor: Edgar Wright

The screenshot shows the Eclipse Java IDE interface with the following details:

- Project Explorer:** Shows the project structure with files like JenaJava, Movies.owl, and persons\_query.txt.
- Code Editor:** Displays the Jena3.java code. The code reads an ontology from a file and lists actors.
- Output Console:** Shows the execution output, listing actors such as Paul Thomas Anderson, Scarlett Johansson, John Travolta, Tom Hanks, Ahmed Ezz, Uma Thurman, Taika Waititi, Quentin Tarantino, and Edgar Wright.

#### 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

```

import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.util.FileManager;

import java.util.Iterator;

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/Movies.owl";

        model.read(fileManager.open(owlFile), null);

        // Prompt user to enter the name of the movie

        String movieName = System.console().readLine("Enter the name of the movie:");

        // Query to retrieve information about the movie

        String queryString = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +

                "PREFIX movies:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>" +

                "SELECT ?year ?country ?genre ?actorName " +

                "WHERE {" +

                " ?movie rdf:type movies:Movies ." +

                " ?movie movies:hasTitle ?title ." +

                " FILTER (str(?title) = \\" + movieName + "\\") ." +

                " OPTIONAL { ?movie movies:hasYear ?year }" +

                " OPTIONAL { ?movie movies:hasCountry ?country }" +

                " OPTIONAL { ?movie movies:hasGenre ?genre }" +

                " ?movie movies:hasActor ?actor ." +

                " ?actor rdf:type movies:Actor." +

                " ?actor movies:hasName ?actorName ." +

                "}";
    }
}

```

```
// Execute the query

try (QueryExecution qexec = QueryExecutionFactory.create(queryString,
model)) {

    ResultSet results = qexec.execSelect();

    if (results.hasNext()) {

        System.out.println("Movie Information:");

        while (results.hasNext()) {

            QuerySolution soln = results.nextSolution();

            RDFNode year = soln.get("year");

            RDFNode country = soln.get("country");

            RDFNode genre = soln.get("genre");

            RDFNode actor = soln.get("actorName");



            if (year != null) {

                System.out.println("Year: " + year.toString());

            }

            if (country != null) {

                System.out.println("Country: " + country.toString());

            }

            if (genre != null) {

                System.out.println("Genre: " + genre.toString());

            }

            if (actor != null) {

                System.out.println("Actor: " + actor.toString());

            }

        }

    } else {

}
```

```

        System.out.println("Movie not found or information not available.");
    }

}

}

}

```

Screenshot of Eclipse IDE showing Java code and a Jena application running in the background.

**Java Code (Jena4.java):**

```

Problems @ Javadoc Declaration Console X
<terminated> Jena4 [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 9, 2024, 3:06:47 AM - 3:06:54 AM) [pid: 1616]
Enter the name of the movie: El Maslaha
Movie Information:
Year: 2012^^http://www.w3.org/2001/XMLSchema#integer
Country: Egypt
Genre: Thriller
Actor: Ahmed Ezz
Year: 2012^^http://www.w3.org/2001/XMLSchema#integer
Country: Egypt
Genre: Action
Actor: Ahmed Ezz

```

**Eclipse IDE (Package Explorer):**

- Project: eclipse\_javaProj
- src:
  - Movies
  - Jena0.java
  - Jena1.java
  - Jena2.java
  - Jena3.java
  - Jena4.java
- Referenced Libraries:
  - Movies.owl
  - owl-rules.txt
  - persons.query.txt

**Java Code (Jena4.java):**

```

eclipse_javaProj - Movies/src/jena4.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X
project Jena0.java Jena1.java persons_query.txt Jena3.java owl-rules.txt Jena4.java X
12 // Load the ontology
13 OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
14 FileManager fileManager = FileManager.get();
15 String owlFile = "data/Movies.owl";
16 model.read(fileManager.open(owlFile), null);
17
18 // Prompt user to enter the name of the movie
19 String movieName = System.console().readline("Enter the name of the movie: ");
20
21 // Query to retrieve information about the movie
22 String queryString = "#PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> +"
23 "PREFIX movies: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#> +"
24 "?WHERE (" +
25 " ?movie rdf:type movies:Movies ."
26 " ?movie movies:hasTitle ?title ."
27 " FILTER (str(?title) = '" + movieName + "') ."
28 " OPTIONAL { ?movie movies:hasYear ?year } ."
29 " OPTIONAL { ?movie movies:hasCountry ?country } ."
30 " OPTIONAL { ?movie movies:hasGenre ?genre } ."
31 " ?movie movies:hasActor ?actor ."
32 " ?actor rdf:type movies:Actor ."
33 " ?actor movies:hasName ?actorName ."
34 " ?actor movies:hasName ?actorName ."
35 ")";
36
37 // Execute the query
38 try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
39   ResultSet results = qexec.execSelect();
40   if (results.hasNext()) {

```

**Java Code (Jena4.java):**

```

Problems @ Javadoc Declaration Console X
<terminated> Jena4 [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 9, 2024, 3:06:47 AM - 3:06:54 AM) [pid: 1616]
Enter the name of the movie: El Maslaha
Movie Information:
Year: 2012^^http://www.w3.org/2001/XMLSchema#integer
Country: Egypt
Genre: Thriller
Actor: Ahmed Ezz
Year: 2012^^http://www.w3.org/2001/XMLSchema#integer
Country: Egypt
Genre: Action
Actor: Ahmed Ezz

```

**Activate Windows**  
Go to Settings to activate Windows.

Screenshot of Eclipse IDE showing Java code and a Jena application running in the background.

**Java Code (Jena4.java):**

```

Problems @ Javadoc Declaration Console X
<terminated> Jena4 [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 9, 2024, 3:10:17 AM - 3:10:24 AM) [pid: 1942]
Enter the name of the movie: Catch me if you can
Movie not found or information not available.

```

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.

Jena5\_tial1.java code:

```

import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.ResultSet;
import org.apache.jena.query.ResultSetFormatter;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.reasoner.Reasoner;

public class Jena5_tial1 {
    public static void main(String[] args)
    {
        Model model = ModelFactory.createDefaultModel();
        model.read( "data/Movies.owl" );

        Reasoner reasoner = new GenericRuleReasoner( Rule.rulesFromURL(
        "data/Jena5_rules.txt" ) );
        InfModel infModel = ModelFactory.createInfModel( reasoner, model );

        readPersonAge(infModel);
    }

    public static void readPersonAge(Model model) {
        String query = "PREFIX ns:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#> +
        "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 . " +
        " ?person ns:isDirectorOf ?movie2 . " +
        "}";
        executeQuery(model, query);
    }
}

```

```

    }

    // Helper method to execute and print query results
    private static void executeQuery(Model model, String queryString) {
        try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
            ResultSet results = qexec.execSelect();
            ResultSetFormatter.out(System.out, results);
        }
    }
}

```

## 6. Jena5\_rules.txt:

```

@prefix ns: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

[rule1: (?per ns:isActorOf ?movie1) (?per ns:isDirectorOf ?movie2) -> (?per rdf:type
ns:ActorDirector) ]

```

S Problems @ Javadoc Declaration Console X  
 [terminated] Jena5\_tial1 [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86\_64\_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 14, 2024, 3:01:29 AM – 3:01:31 AM) [pid: 12456]

person	movie1	movie2
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Taika_Waititi>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Jojo_Rabbit>	<http://www.semanticweb.org/hp/onto:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Taika_Waititi>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Jojo_Rabbit>	<http://www.semanticweb.org/hp/onto:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill>	<http://www.semanticweb.org/hp/onto:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill>	<http://www.semanticweb.org/hp/onto:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction>	<http://www.semanticweb.org/hp/onto:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction>	<http://www.semanticweb.org/hp/onto:

movie1	movie2
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Jojo_Rabbit>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Thor:Ragnarok>
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Jojo_Rabbit>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Jojo_Rabbit>
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction>
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill>
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction>
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction>	<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill>

```

import java.io.InputStream;

import org.apache.jena.query.*;

import org.apache.jena.rdf.model.Model;

import org.apache.jena.rdf.model.ModelFactory;

import org.apache.jena.util.FileManager;

```

```
public class Jena5 {  
  
    public static void main(String[] args) {  
  
        // Load the ontology  
  
        Model model = ModelFactory.createDefaultModel();  
  
        FileManager fileManager = FileManager.get();  
  
        String owlFile = "data/Movies.owl"; // Replace with the actual filename  
  
        model.read(fileManager.open(owlFile), null);  
  
  
        // Read the SPARQL query from file  
  
        String queryFile = "data/actor_rules.txt";  
  
        String queryString = readQueryFromFile(queryFile);  
  
  
        // Execute the query and create a new model with the constructed triples  
  
        Model resultModel = ModelFactory.createDefaultModel();  
  
        try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {  
  
            resultModel = qexec.execConstruct();  
  
        }  
  
  
        // Extract and display the names of the persons who are both actors and directors  
  
        System.out.println("Persons who are both actors and directors:");  
  
  
        resultModel.listObjectsOfProperty(resultModel.createProperty("http://www.semanticweb.org/h  
p/ontologies/2024/3/Movies#hasName"))  
  
            .forEachRemaining(name -> System.out.println(name.toString()));  
  
    }  
  
  
    // Helper method to read the query from file  
  
    private static String readQueryFromFile(String queryFile) {
```

```
StringBuilder sb = new StringBuilder();

try (InputStream inputStream = FileManager.get().open(queryFile)) {

    if (inputStream != null) {

        String line;

        java.io.BufferedReader reader = new java.io.BufferedReader(
            new java.io.InputStreamReader(inputStream, "UTF-8"));

        while ((line = reader.readLine()) != null) {

            sb.append(line).append("\n");

        }

    }

} catch (java.io.IOException e) {

    e.printStackTrace();

}

return sb.toString();

}
```

```
Problems @ Javadoc Declaration Console X
<terminated> Jena5 [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.jst.j.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 9, 2024, 3:32:37 AM – 3:32:39 AM) [pid: 1440]
Persons who are both actors and directors:
Taika Waititi
Tom Hanks
Quentin Tarantino
Paul Thomas Anderson
Edgar Wright
```

- **actor\_rules.txt** file:

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

PREFIX movies: <<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>>

## CONSTRUCT {

```
?person movies:hasName ?name .  
}
```

```

WHERE {

?person rdf:type movies:Actor .

?person rdf:type movies:Director .

?person movies:hasName ?name .

}

```

```

eclipse.javaProj - Movies/src/jena5Java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X Jena0Java Movies.owl Jena2.java persons_query.txt Jena3.java Jena4.java Jena5Java X actor_rules.txt
Movies > JRE System Library [jre]
src > (default package)
Jena0Java
Jena2Java
Jena3Java
Jena4Java
Jena5Java
Referenced Libraries
data
actor_rules.txt
Movies.owl
persons_query.txt

public class Jena5 {
    public static void main(String[] args) {
        // Load the ontology
        Model model = ModelFactory.createDefaultModel();
        FileManager fileManager = FileManager.get();
        String owlfile = "data/Movies.owl";
        model.read(fileManager.open(owlfile), null);

        // Read the SPARQL query from file
        String queryfile = "data/actor_rules.txt";
        String queryString = readQueryFromFile(queryfile);

        // Execute the query and create a new model with the constructed triples
        Model resultModel = ModelFactory.createDefaultModel();
        try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
            resultModel = qexec.execConstruct();
        }

        // Extract and display the names of the persons who are both actors and directors
        System.out.println("Persons who are both actors and directors:");
        resultModel.listObjectsOfProperty(resultModel.createProperty("http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasName"))
            .forEachRemaining(name -> System.out.println(name.toString()));
    }

    // Helper method to read the query from file
    private static String readQueryFromFile(String queryFile) {
        StringBuilder sb = new StringBuilder();
        try (InputStream inputStream = fileManager.get().open(queryFile)) {
            if (inputStream != null) {
                ...
            }
        }
    }
}

<terminated> Jena5 [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.jst\openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 9, 2024, 3:32:37 AM - 3:32:39 AM) [pid: 1440]
Persons who are both actors and directors:
Taika Waititi
Tom Hanks
Quentin Tarantino
Paul Thomas Anderson
Edgar Wright

```

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

- Jena6\_trial1 code:

```

import org.apache.jena.query.*;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
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_trial1 {
    public static void main(String[] args) {
        // Load the ontology
        Model model = ModelFactory.createDefaultModel();
        FileManager fileManager = FileManager.get();
        String owlFile = "data/Movies.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);

        // Define rules
        String rule1 = "[ruleHasTwoRoles: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasActor> ?person) (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasDirector> ?person) ->
(?person <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#HasTwoRoles>
?movie)]";
        String rule2 = "[ruleMoviesDirectedByQT: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasDirector>
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino>) ->
(?movie <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#DirectedByQT>
?movie)]";
        String rule3 = "[ruleMoviesInEgypt: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasCountry>
\"Egypt\"^^xsd:string) (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasActor> ?actor) -> (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ActorName> ?actor)]";
        String rule4 = "[ruleMoviesInEgypt: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasGenre>
\"Thriller\"^^xsd:string) (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasActor> ?actor) -> (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ThrillerActor> ?actor)]";
        String rule5 = "[ruleMoviesInEgypt: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasGenre>
\"Romance\"^^xsd:string) (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasActor> ?actor) -> (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#RomanceActor> ?actor)]";

        // Create a reasoner with the rules
        Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rule1 + " " + rule2 + " "
+ rule3+ " "+rule4+ " " + rule5));

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

```

```

// Execute queries
askForTwoRoles(infModel);
askMoviesDirectedByQuentinTarantino(infModel);
askMoviesInCountry(infModel, "Egypt");
askMoviesInGenre(infModel, "Thriller");
askMoviesInGenreRomance(infModel, "Romance");
}

// Rule 1: Persons Who Are Actors And Directors
private static void askForTwoRoles(Model model) {
    System.out.println("Persons who are actors and directors:");
    String queryString = "SELECT ?person WHERE {?person
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#HasTwoRoles> ?movie}";
    executeAndPrintQuery(model, queryString);
}

// Rule 2: Movies Directed by Quentin Tarantino
private static void askMoviesDirectedByQuentinTarantino(Model model) {
    System.out.println("Movies directed by Quentin Tarantino:");
    String queryString = "SELECT ?movieTitle WHERE {?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#DirectedByQT> ?movie .
?movie <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasTitle>
?movieTitle}";
    executeAndPrintQuery(model, queryString);
}

// Rule 3: Movies Released in a Specific Country with Directors' Names
private static void askMoviesInCountry(Model model, String country) {
    System.out.println("Movies released in " + country + " along with their actor:");
    String queryString = "SELECT ?movieTitle ?actor WHERE {?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ActorName> ?actor . ?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasTitle> ?movieTitle}";
    executeAndPrintQuery(model, queryString);
}

// Rule 4: Movies with Thriller genre
private static void askMoviesInGenre(Model model, String genre) {
    System.out.println("Movies have thriller genre " + genre + " along with their actor:");
    String queryString = "SELECT ?movieTitle ?actor WHERE {?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ThrillerActor> ?actor . ?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasTitle> ?movieTitle}";
    executeAndPrintQuery(model, queryString);
}

// Rule 4: Movies with Romance genre

```

```

private static void askMoviesInGenreRomance(Model model, String genre1) {
    System.out.println("Movies have thriller genre " + genre1 + " along with their actor:");
    String queryString = "SELECT ?movieTitle ?actor WHERE {?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#RomanceActor> ?actor .
?movie <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasTitle>
?movieTitle}";
    executeAndPrintQuery(model, queryString);
}

// Helper method to execute and print query results
private static void executeAndPrintQuery(Model model, String queryString) {
    try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
        ResultSet results = qexec.execSelect();
        ResultSetFormatter.out(System.out, results);
    }
}

```

- Output:

The screenshot shows two terminal windows in the Eclipse IDE. Both windows are titled 'Console' and show the output of SPARQL queries.

**Top Terminal Window Output:**

```

terminated> Jena6_trial1 [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.jst\openjdkhotspot\jre\full\win32\x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 13, 2024, 6:05:50 PM – 6:05:55 PM) [pid: 8244]
Persons who are actors and directors:
-----
| person
=====
| <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino>
| <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino>
| <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Taika_Waititi>
|
Movies directed by Quentin Tarantino:
-----
| movieTitle
=====
| "Pulp Fiction"
| "Kill Bill (volume1)"
|
Movies released in Egypt along with their actor:
-----
| movieTitle | actor
=====
| "El Maslaha" | <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Ahmed_Ezz>
| "El Maman" | <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Ahmed_Ezz>
| "Africano" | <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Mona_Zaki>

```

**Bottom Terminal Window Output:**

```

terminated> Jena6_trial1 [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.jst\openjdkhotspot\jre\full\win32\x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 13, 2024, 6:30:26 PM – 6:30:28 PM) [pid: 20984]
-----
| movieTitle | actor
=====
| "Forrest Gump" | <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Tom_Hanks>

```

- Created Jena6\_rules.java using inferred rules:
- The rule finds actors that are directors at the same time and display the movies they acted and directed in with new property called “HasTwoRoles”

```
import org.apache.jena.ontology.OntModel;
```

```
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.*;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.util.FileManager;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.rdf.model.Statement;
import org.apache.jena.rdf.model StmtIterator;
import org.apache.jena.reasoner.Reasoner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Iterator;

public class Jena6_rules {
    public static void main(String[] args)
    {
        Model model = ModelFactory.createDefaultModel();
        model.read( "data/Movies.owl" );

        Reasoner reasoner = new GenericRuleReasoner( Rule.rulesFromURL( "data/rul1.txt" ) );

        InfModel infModel = ModelFactory.createInfModel( reasoner, model );
        StmtIterator it = infModel.listStatements();

        while ( it.hasNext() )
        {
            Statement stmt = it.nextStatement();

            Resource subject = stmt.getSubject();
            Property predicate = stmt.getPredicate();
            RDFNode object = stmt.getObject();
            System.out.println( subject.toString() + " " + predicate.toString() + " " + object.toString()
);
        }
    }
}
```

- rul1.txt:

```
@prefix ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

[ruleHasTwoRoles: (?s ont:hasActor ?c) (?b ont:hasDirector ?c) -> (?c ont:HasTwoRoles ?s) (?c ont:HasTwoRoles ?b)]
```

- Output:

```
<terminated> Jena6_rules [Java Application] C:\Users\HP.p2\pool\plugins\org.eclipse.jst.java.core\openjdk\hotspot\jre\full\win32\x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 13, 2024, 3:19:32 AM – 3:19:34 AM) [pid: 30716]
http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Taika_Waititi http://www.semanticweb.org/hp/ontologies/2024/3/Movies#HasTwoRoles http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Thor_Ragnarok
http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Taika_Waititi http://www.semanticweb.org/hp/ontologies/2024/3/Movies#HasTwoRoles http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Jojo_Rabbit
http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino http://www.semanticweb.org/hp/ontologies/2024/3/Movies#HasTwoRoles http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill
http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino http://www.semanticweb.org/hp/ontologies/2024/3/Movies#HasTwoRoles http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction
```

- Jena7 is to display actors below 51 using rule:
- Jena7 code:

```
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.query.ResultSetFormatter;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.*;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.util.FileManager;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.rdf.model.Statement;
import org.apache.jena.rdf.model.StmtIterator;
import org.apache.jena.reasoner.Reasoner;
import java.io.InputStream;
import java.io.FileNotFoundException;
import java.util.Iterator;

public class Jena7 {
    public static void main(String[] args)
    {
        Model model = ModelFactory.createDefaultModel();
        model.read( "data/Movies.owl" );

        Reasoner reasoner = new GenericRuleReasoner( Rule.rulesFromURL(
        "data/Jena7_rules.txt" ) );
        InfModel infModel = ModelFactory.createInfModel( reasoner, model );
```

```

        readPersonAge(infModel);
    }

    public static void readPersonAge(Model model) {
        String query = "PREFIX ns:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#> +
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> +
SELECT ?person ?age +
WHERE { ?person rdf:type ns:PersonAge .
?person ns:hasAge ?age .
} ;
executeQuery(model, query);
}

// Helper method to execute and print query results
private static void executeQuery(Model model, String queryString) {
    try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
        ResultSet results = qexec.execSelect();
        ResultSetFormatter.out(System.out, results);
    }
}
}

```

- Jena7\_rule.txt:

```

@prefix ns: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

[rule1: (?per rdf:type ns:Persons) (?per ns:hasAge ?age) lessThan(?age, 51)-> (?per
rdf:type ns:PersonAge) ]

```

- Output:

The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the following table:

person	age
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Edgar_Wright>	48
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Taika_Waititi>	47
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Mona_Zaki>	47
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Uma_Thurman>	43
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Scarlett_Johansson>	39

## Test Cases:

<https://jena.apache.org/documentation/inference/>

A suite of test cases designed to evaluate the correctness and completeness of the ontology. These test cases may cover various aspects such as ontology consistency, inference correctness, and query performance.

## Inference Correctness:

Inference correctness focuses on assessing whether the ontology's reasoning mechanisms produce accurate and valid results. Test cases in this category evaluate the ontology's ability to infer new knowledge based on existing data and logical rules. These tests often involve scenarios where the ontology needs to deduce implicit information or make logical conclusions from explicit assertions. Examples include verifying the correctness of inferred relationships between entities or validating the outcomes of logical entailment based on ontology axioms.

The most common reasoner operation which can't be exposed through additional triples in the inference model is that of validation. Typically the ontology languages used with the semantic web allow constraints to be expressed, the validation interface is used to detect when such constraints are violated by some data set.

A simple but typical example is that of datatype ranges in RDFS. RDFS allows us to specify the range of a property as lying within the value space of some datatype. If an RDF statement asserts an object value for that property which lies outside the given value space there is an inconsistency.

To test for inconsistencies with a data set using a reasoner we use the `InfModel.validate()` interface. This performs a global check across the schema and instance data looking for inconsistencies. The result is a `ValidityReport` object which comprises a simple pass/fail flag (`ValidityReport.isValid()`) together with a list of specific reports (instances of the `ValidityReport.Report` interface) which detail any detected inconsistencies. At a minimum the individual reports should be printable descriptions of the problem but they can also contain an arbitrary reasoner-specific object which can be used to pass additional information which can be used for programmatic handling of the violations.

```
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 = "data/Movies.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());
            }
        }
    }
}

```



- Possible outcome if there is inconsistencies:

The file testing/reasoners/rdfs/dttest2.nt declares a property bar with range xsd:integer and attaches a bar value to some resource with the value "25.5"^^xsd:decimal. If we run the above sample code on this file we see:

*Conflicts*

- Error (dtRange): Property <http://www.hpl.hp.com/semweb/2003/eg#bar> has a typed range Datatype [<http://www.w3.org/2001/XMLSchema#integer> -> class java.math.BigInteger] that is not compatible with 25.5 <http://www.w3.org/2001/XMLSchema#decimal>

- Made a test suite using JUNIT Tes1\_Inference:

eclipse\_javaProj - Movies/src/Testing/Test1\_Inference.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit X

Finished after 1.474 seconds

Runs: 4/4 Errors: 0 Failures: 0

```

5 import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
6 import org.apache.jena.reasoner.rulesys.Rule;
7 import org.apache.jena.reasoner.Reasoner;
8 import org.apache.jena.ontology.OntModel;
9 import org.apache.jena.ontology.OntProperty;
10 import org.apache.jena.ontology.OntResource;
11 import org.apache.jena.rdf.model.ModelFactory;
12 import org.apache.jena.util.iterator.ExtendedIterator;
13 import org.junit.Test;
14 import org.apache.jena.reasoner.*;
15 import org.apache.jena.rdf.model.InfModel;
16
17
18 public class Test1_Inference {
19     @Test
20     public void testSubclassRelationship() {
21         // Load the ontology
22         OntModel model = ModelFactory.createOntologyModel();
23         model.read("data/Movies.owl");
24
25         // Perform inference and check if there exists a subclass relationship
26         boolean isSubclass = model.getOntClass("http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Persons")
27             .hasSubClass(model.getOntClass("http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Actor"));
28
29         // Assert that the subclass relationship exists
30         assertTrue("Actor should be a subclass of Persons", isSubclass);
31     }
32     @Test
33     public void testPropertyInference() {
34         // Load the ontology
35         OntModel model = ModelFactory.createOntologyModel();
36         model.read("data/Movies.owl");
37

```

Failure Trace

Problems Javadoc Declaration Console X

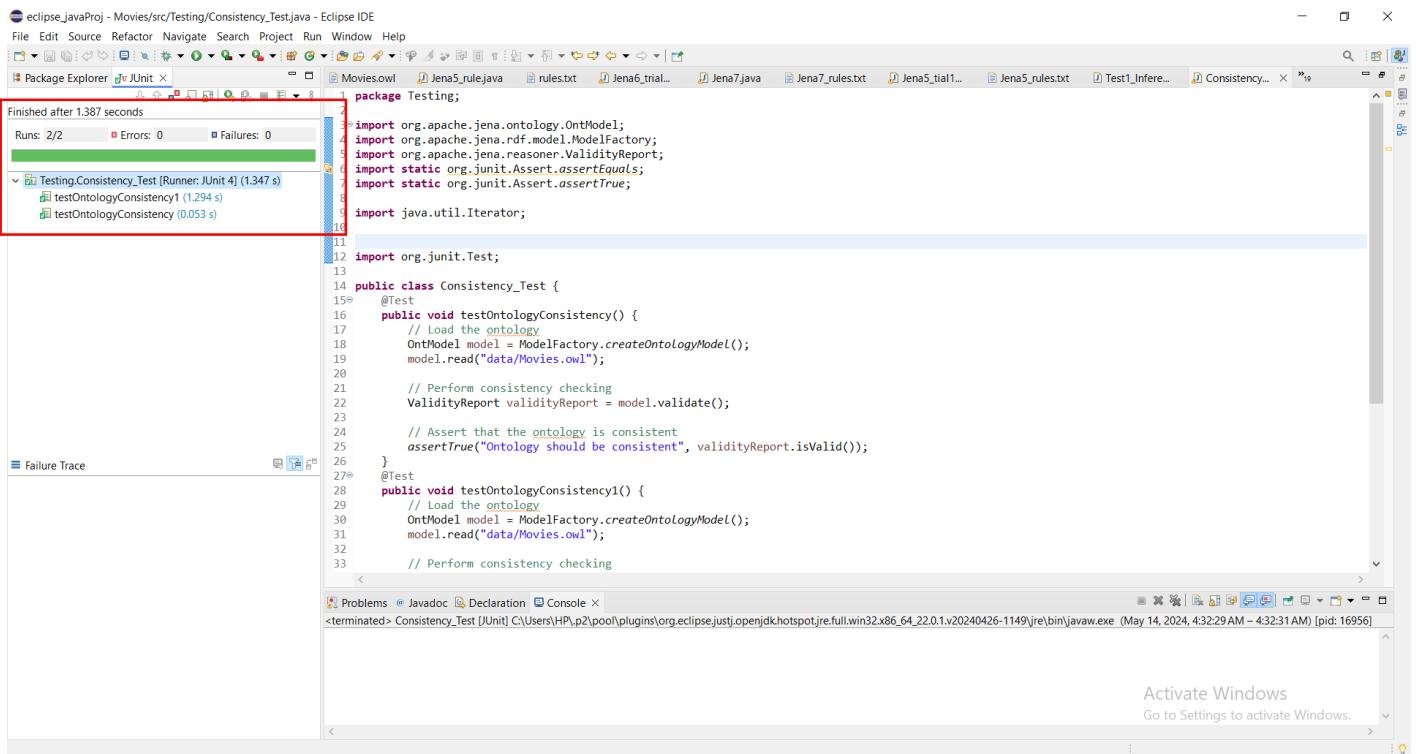
<terminated> Test1\_Inference [JUnit] C:\Users\HP\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86\_64\_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 14, 2024, 4:33:46 AM - 4:33:49 AM) [pid: 16620]

Activate Windows  
Go to Settings to activate Windows.

## Ontology Consistency:

This aspect ensures that the ontology maintains internal coherence and integrity. Test cases under this category typically involve verifying that the relationships, properties, and constraints defined within the ontology are consistent and adhere to specified rules and axioms. For instance, test cases may check for inconsistencies in data representation, such as conflicting statements about the same entity, or violations of defined constraints.

- Created Test suite called `Consistency_Test`:



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows files like `Movies.owl`, `Jena5_rulejava`, `rules.txt`, etc.
- JUnit View:** Contains the following information:
  - Finished after 1.387 seconds
  - Runs: 2/2 Errors: 0 Failures: 0
  - Testing.Consistency\_Test [Runner: JUnit 4] (1.347 s)
    - testOntologyConsistency1 (1.294 s)
    - testOntologyConsistency (0.053 s)
- Code Editor:** Displays the Java code for `Consistency_Test`.
- Bottom Status Bar:** Shows the message: "Activate Windows Go to Settings to activate Windows."

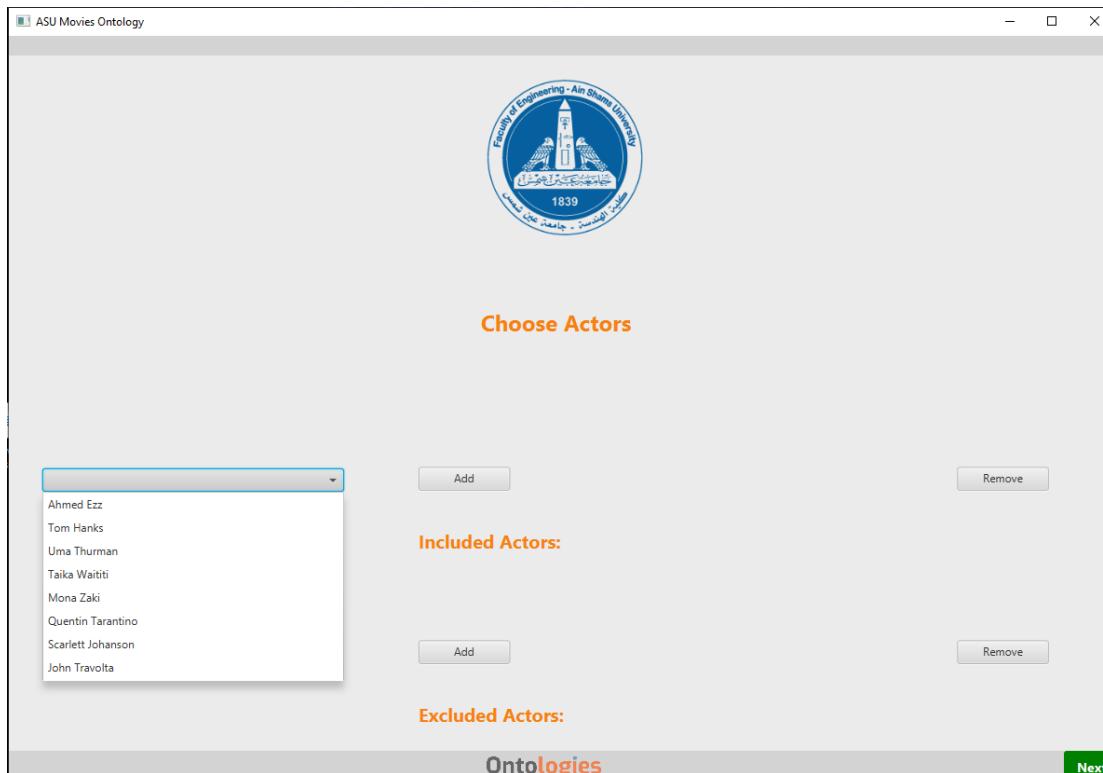
## Query Performance:

Query performance evaluates the efficiency and effectiveness of querying operations on the ontology. Test cases under this aspect assess the ontology's responsiveness and scalability when processing queries of varying complexity and size. This includes measuring the speed of query execution, evaluating resource utilization (such as memory and CPU usage), and assessing the ontology's ability to handle concurrent or parallel queries. Test scenarios may involve benchmarking query response times, stress testing the system with a large volume of queries, and analyzing performance metrics under different load conditions.

## Snapshots of the Interface:

### 1<sup>st</sup>: Choose the actors to be included\excluded:

- The application queries the ontology for all actor names allowing the user to choose the actors to be included\excluded through an interactive GUI .



- Include “Scarlett Johansen”

**ASU Movies Ontology**



### Choose Actors

**Included Actors:**

Ahmed Ezz	Add	Remove
Tom Hanks		
Uma Thurman		
Taika Waititi		
Mona Zaki		
Quentin Tarantino		
<b>Scarlett Johanson</b>	Add	Remove

**Excluded Actors:**

**Ontologies** **Next**

**ASU Movies Ontology**



### Choose Actors

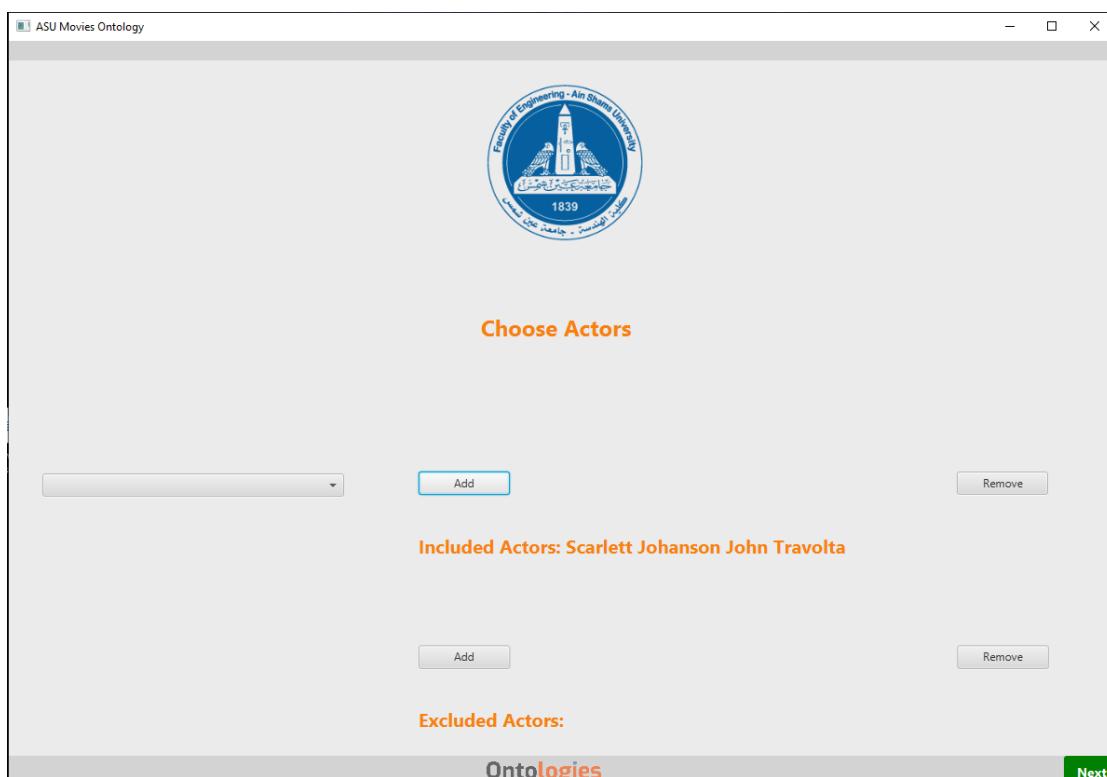
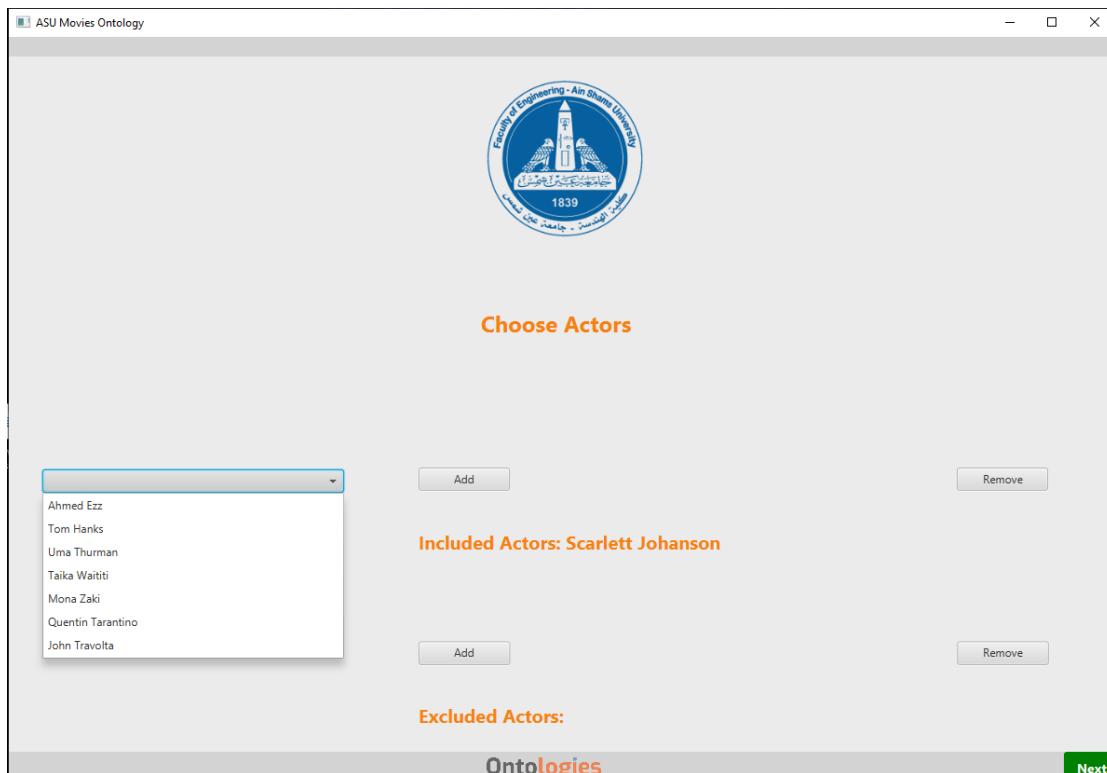
**Included Actors: Scarlett Johanson**

Add	Remove
-----	--------

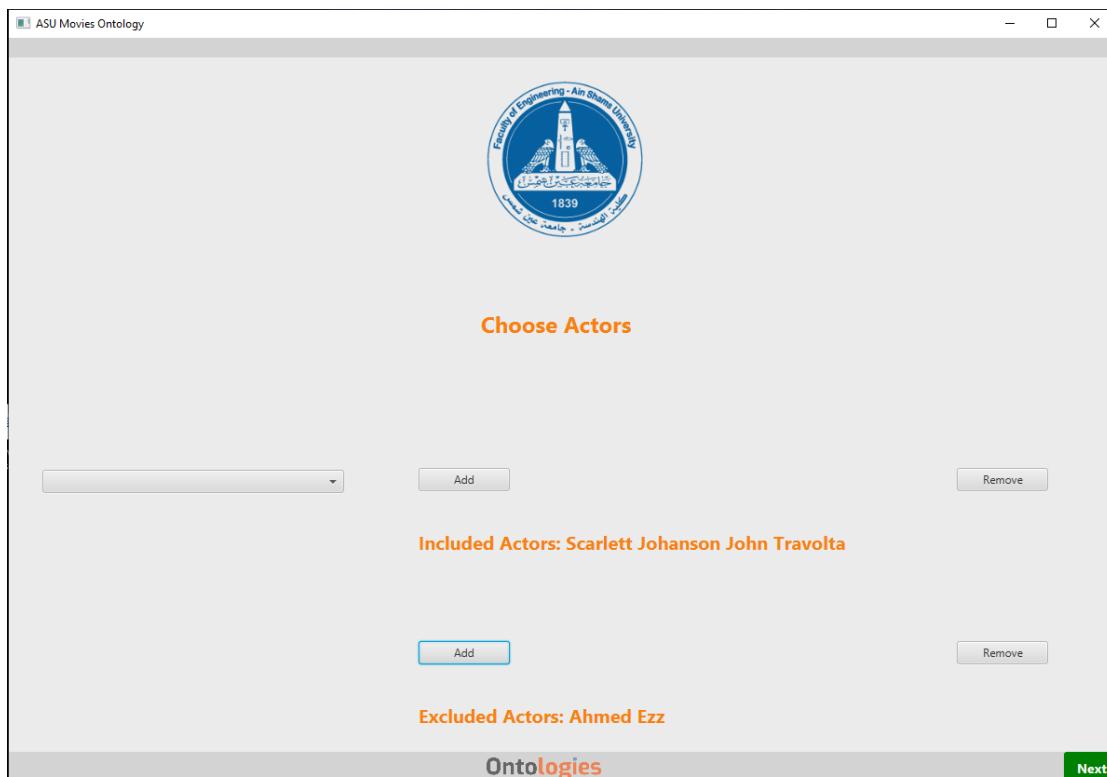
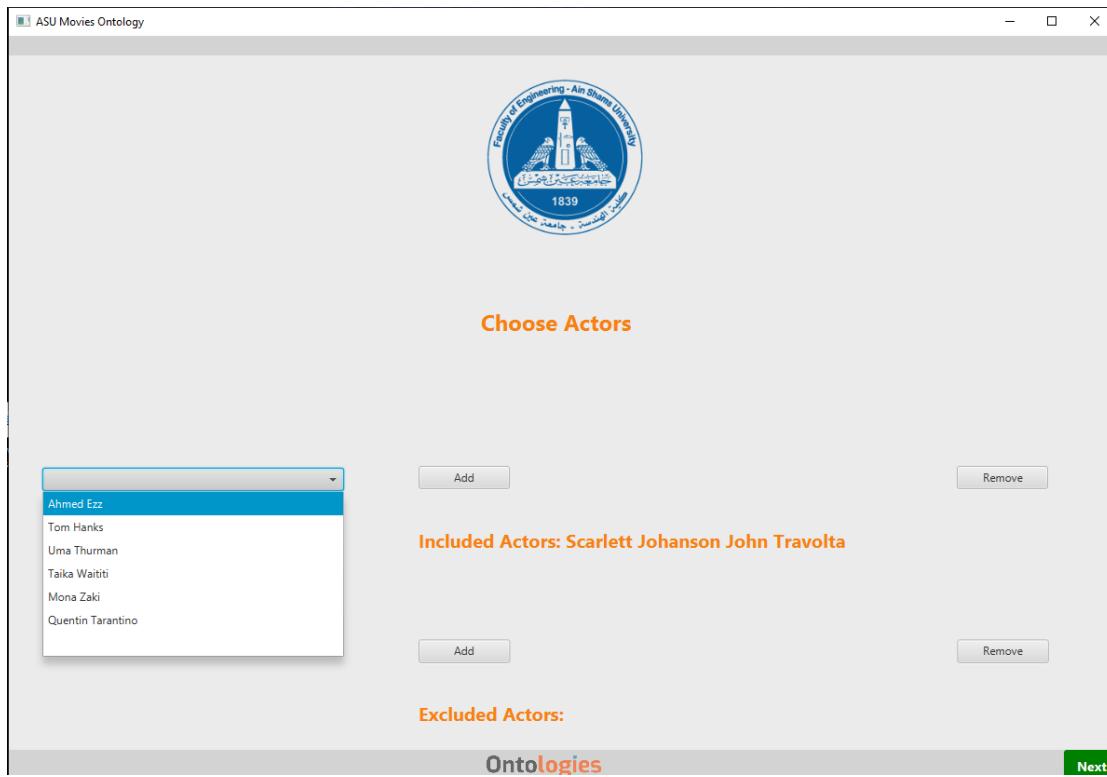
**Excluded Actors:**

**Ontologies** **Next**

- Include “John Travolta”



- Exclude “Ahmed Ezz”



## 2<sup>nd</sup>: Press Next and include/exclude directors:

- Include “Quentin Tarantino”

ASU Movies Ontology



Choose Directors

Included Directors:

Excluded Directors:

Back **Ontologies** Next

ASU Movies Ontology



Choose Directors

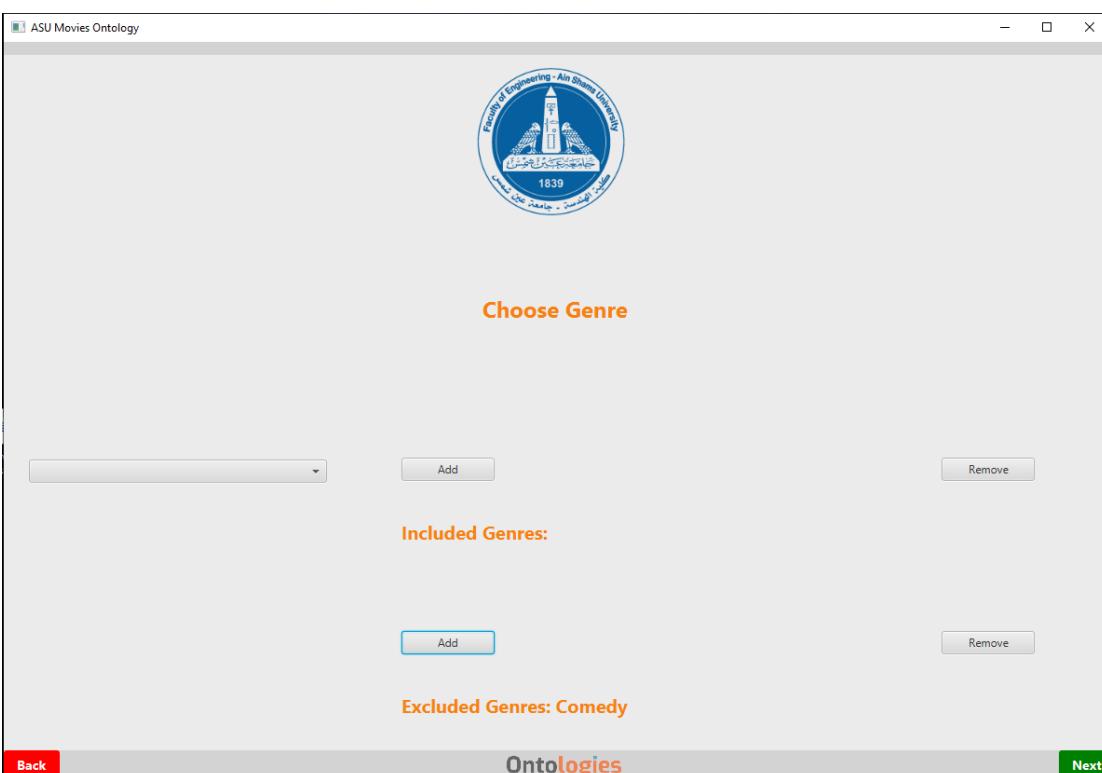
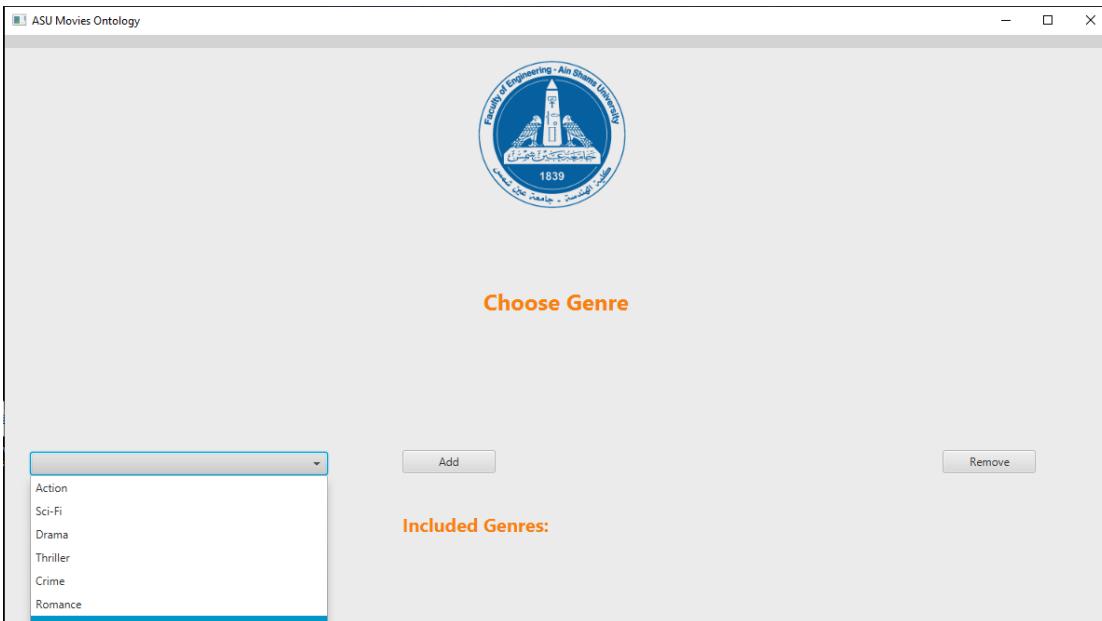
Included Directors: Quentin Tarantino

Excluded Directors:

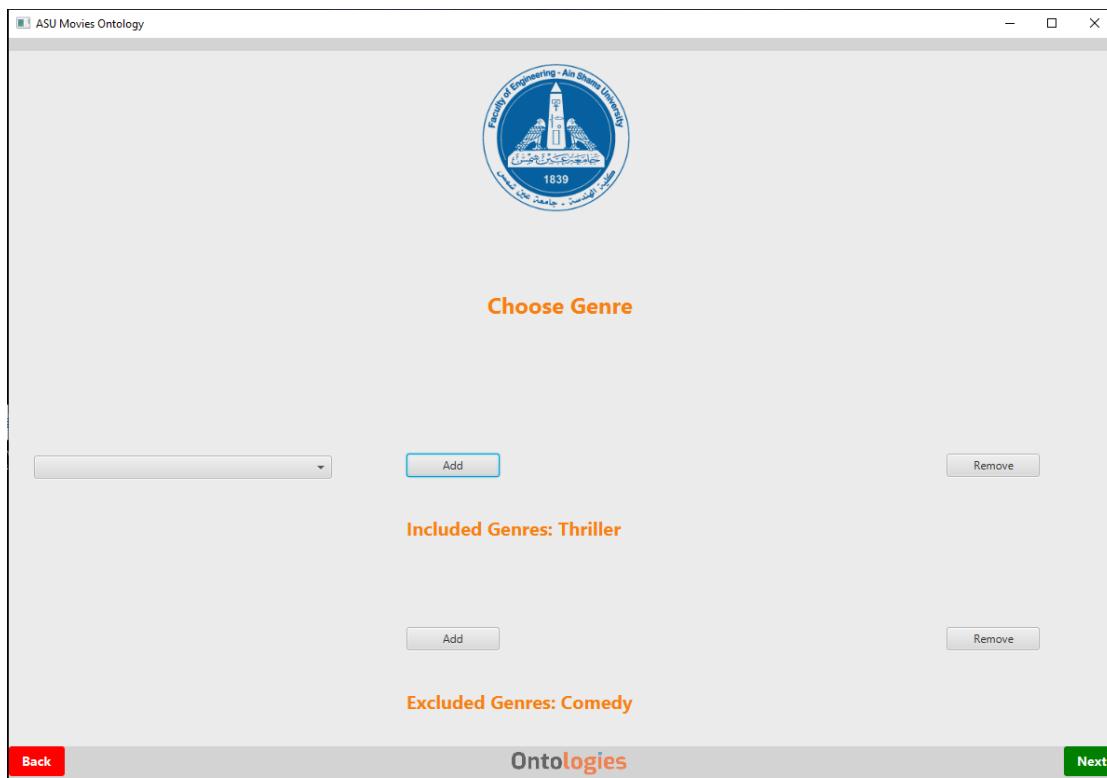
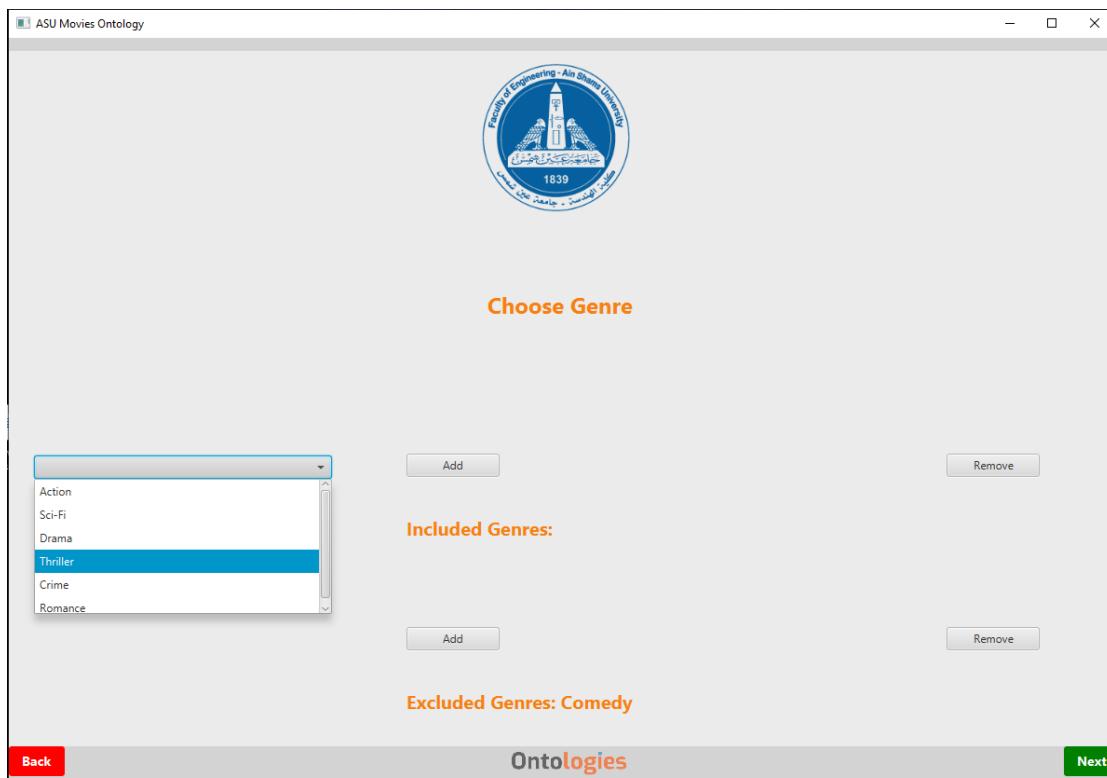
Back **Ontologies** Next

### 3<sup>rd</sup>: Press Next and include/exclude genres:

- Exclude “Comedy”



- Include “Thriller”



4<sup>th</sup>: Press Next and Search to display the query result:

The screenshot shows a web browser window with the title "ASU Movies Ontology". At the top center is the logo of the Faculty of Engineering at Ain Shams University, featuring a circular emblem with two pyramids and the year 1839. Below the logo, the text "Search Results" is displayed in orange. A large white rectangular area contains the search results: "Movie Title: Pulp Fiction Release Year: 1994". To the right of this text is a blue-bordered "Search" button. At the bottom of the page, there is a grey footer bar with a red "Back" button on the left and an orange "Ontologies" button on the right.

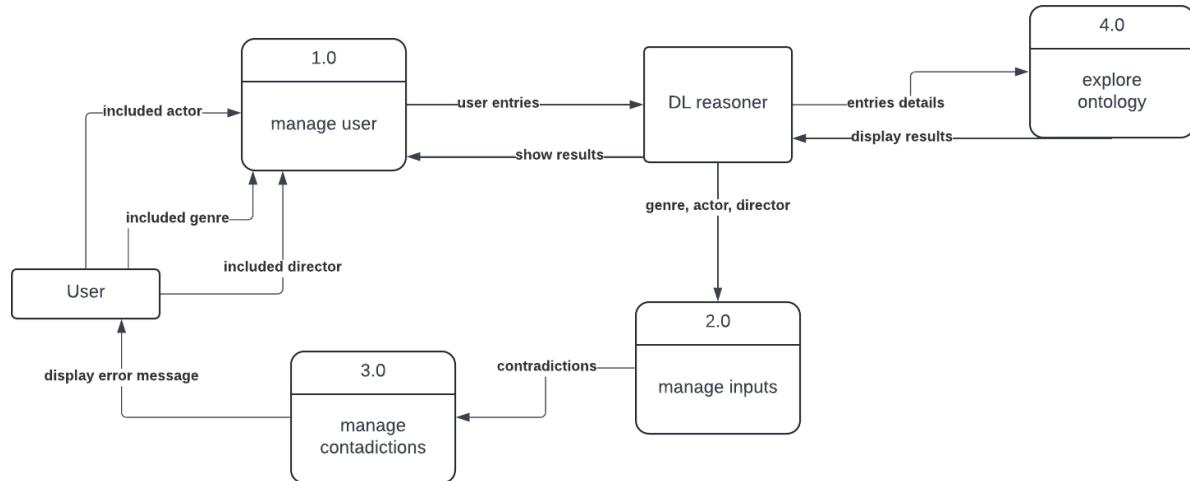
## Bonus: What does the query string look like?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX eg: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT DISTINCT ?movieTitle ?releaseYear
WHERE {
  ?movie rdf:type eg:Movies .
  ?movie eg:hasTitle ?movieTitle .
  ?movie eg:hasYear ?releaseYear .
  ?movie eg:hasActor ?actor .
  ?movie eg:hasDirector ?director .
  ?movie eg:hasGenre ?genre .
  FILTER( ?genre = "Thriller" )
  FILTER(!(?genre = "Comedy"))
  ?director eg:hasName ?directorName .
  FILTER( ?directorName = "Quentin Tarantino" )
  ?actor eg:hasName ?actorName .
  FILTER( ?actorName = "Scarlett Johanson" || ?actorName = "John Travolta" )
  ?actor eg:hasName ?actorName .
  FILTER(!(?actorName = "Ahmed Ezz"))
}
```

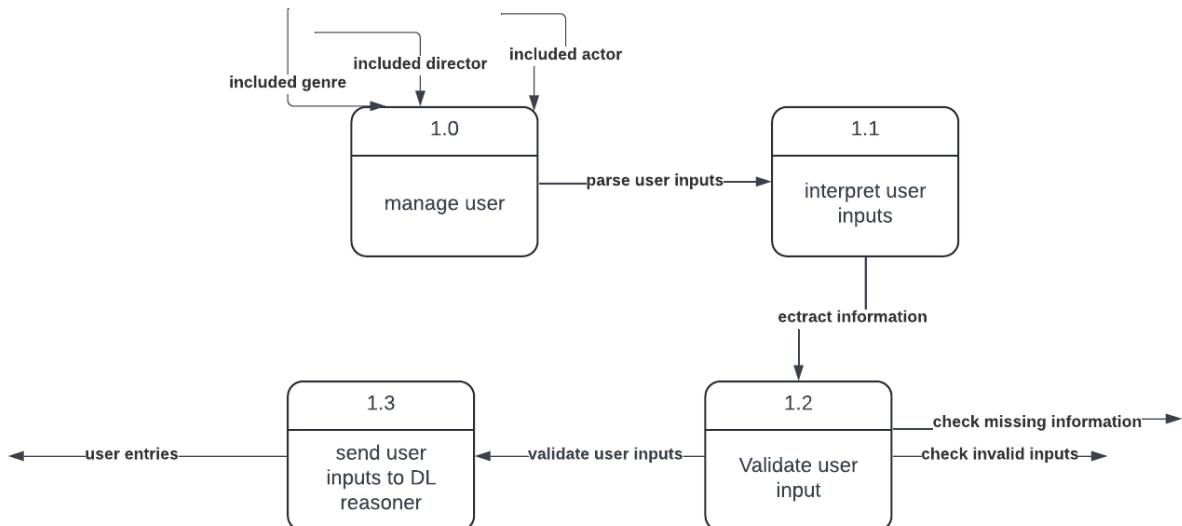
## DFD (Data Flow Diagram):

A diagram illustrating the flow of data within the system, showing how information is input, processed, and output.

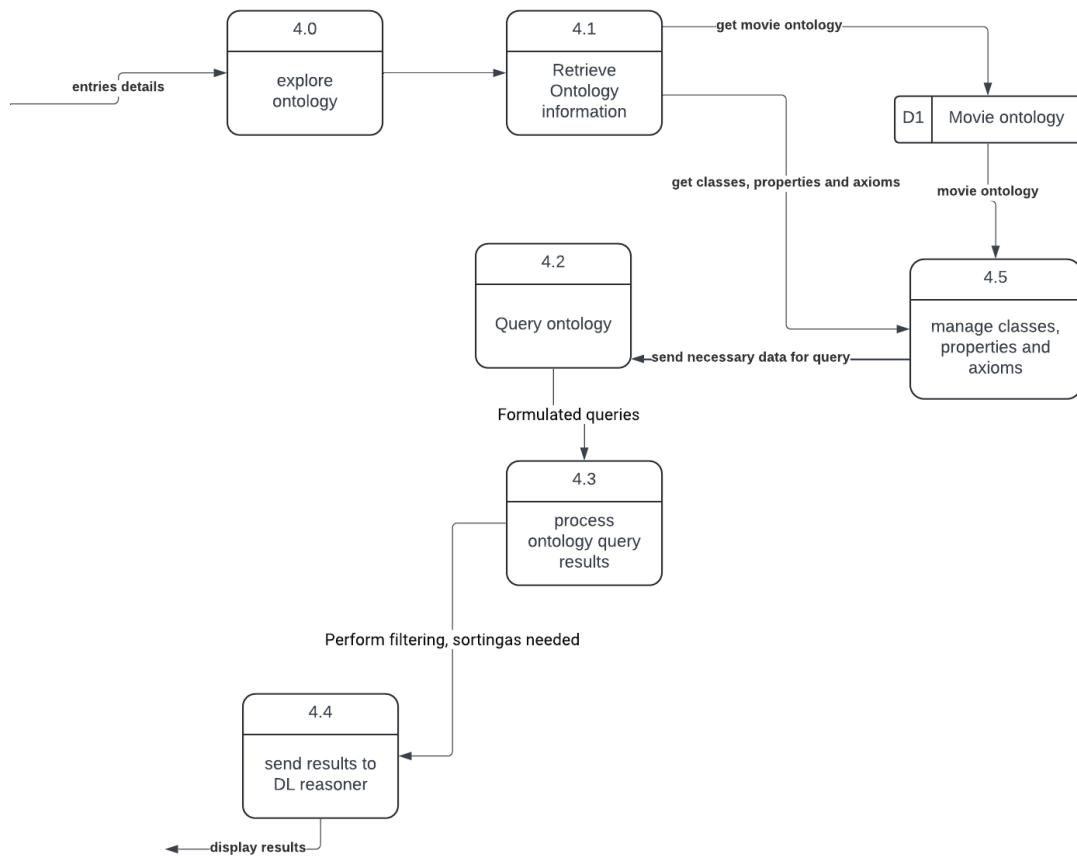
### The whole system:



### Manage user component:

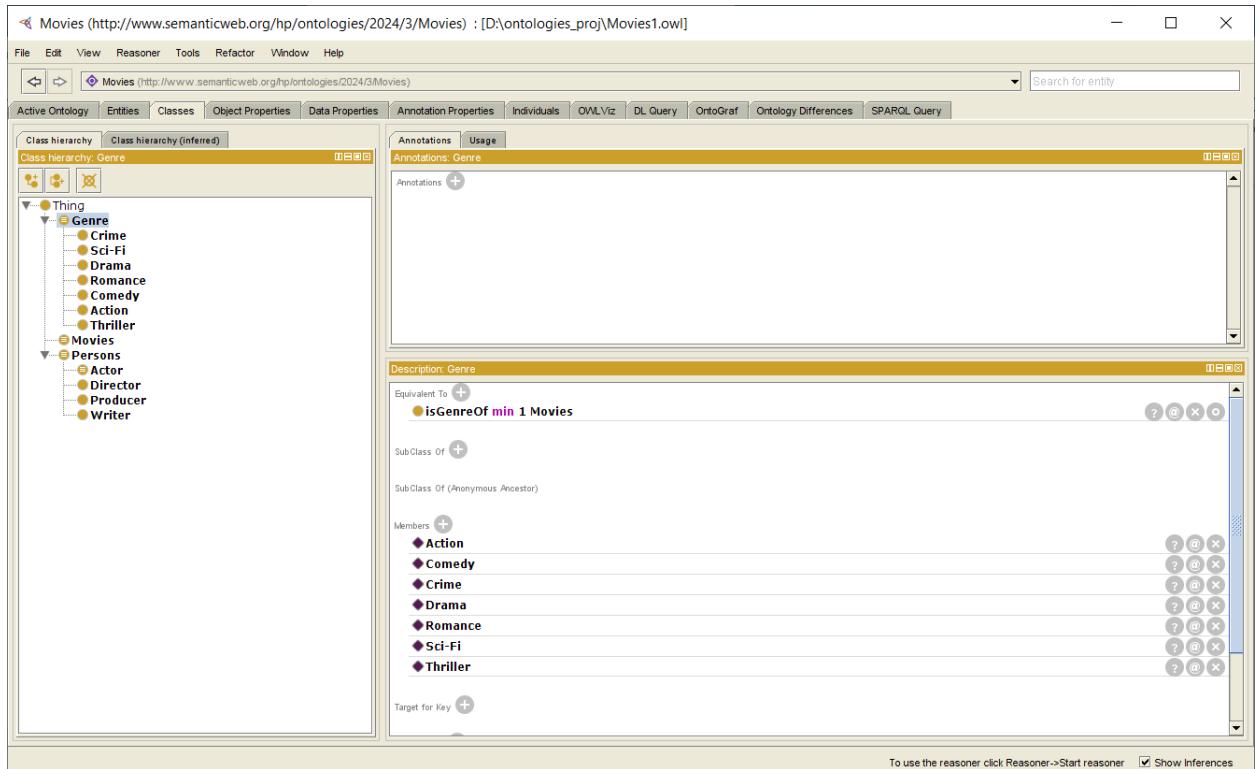


## Explore ontology Component:



## Additional Part:

- Concerning this part we made an adjustment: added Genre Class to the ontology and Action, Crime, Comedy, Drama, Romance, Thriller and Sci-Fi as subclasses
- Made it disjoint with Persons and Movies
- Created individuals of: Action, Crime, Comedy, Drama, Romance, Thriller and Sci-Fi.
- Created object property hasGenre, isGenreOf to connect between Movies and Genre classes



Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies1.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: hasGenre

Annotations (+)

Characteristics: hasG

- Functional
- Inverse functional
- Transitive
- Symmetric
- Asymmetric
- Reflexive
- Irreflexive

Description: hasGenre

Inverse Of (+) **isGenreOf**

Domains (Intersection) (+) **Movies**

Ranges (Intersection) (+) **Genre**

Disjoint With (+)

SuperProperty Of (Chain) (+)

To use the reasoner click Reasoner->Start reasoner  Show Inferences

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies1.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: Action

Annotations (+)

Description: Action

Types (+)

- Action
- Genre

Same Individual As (+)

Different Individuals (+)

Property assertions: Action

Object property assertions (+)

- hasGenre Thor\_Ragnarok**
- isGenreOf El\_Mamar**
- isGenreOf El\_Maslah**
- isGenreOf Africano**
- hasGenre Kill\_Bill**

Data property assertions (+)

Negative object property assertions (+)

Negative data property assertions (+)

To use the reasoner click Reasoner->Start reasoner  Show Inferences

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies1.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: Genre

Annotations: +

Description: Genre

Equivalent To +

- isGenreOf min 1 Movies**

SubClass Of +

SubClass Of (Anonymous Ancestor)

Members +

- Action
- Comedy
- Crime
- Drama
- Romance
- Sci-Fi
- Thriller

Target for Key +

To use the reasoner click Reasoner->Start reasoner  Show Inferences

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies1.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: Movies

Annotations: +

Description: Movies

Equivalent To +

- hasActor min 1 Actor**
- hasGenre min 1 Genre**
- hasLanguage min 1 anyURI
- hasProducer min 1 Producer
- hasCountry min 1 string
- hasTitle exactly 1 string
- hasYear exactly 1 integer
- hasWriter min 1 Writer
- hasDirector min 1 Director

SubClass Of +

SubClass Of (Anonymous Ancestor)

Members +

- Africano
- Baby\_Driver

To use the reasoner click Reasoner->Start reasoner  Show Inferences

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies1.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Data property hierarchy: GenreName

Annotations: GenreName

Characteristics: GenreName

Description: GenreName

Ranges: string

To use the reasoner click Reasoner->Start reasoner  Show Inferences

Movies (<http://www.semanticweb.org/hp/ontologies/2024/3/Movies>) : [D:\ontologies\_proj\Movies1.owl]

File Edit View Reasoner Tools Refactor Window Help

Annotations Usage

Annotations: Comedy

Description: Comedy

Object property assertions: Comedy

Data property assertions: Comedy

To use the reasoner click Reasoner->Start reasoner  Show Inferences

- All rul1.txt:

```
@prefix ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
[ruleHasTwoRoles: (?s ont:hasActor ?c) (?b ont:hasDirector ?c) -> (?c
ont:ActorAndWriter ?s) (?c ont:ActorAndDirector ?b)]
[ruleActorAndWriter: (?s ont:hasActor ?c) (?b ont:hasWriter ?c) -> (?c
ont:ActorAndWriter ?s) (?c ont:ActorAndWriter ?b)]
[ruleGenreClassification: (?movie ont:hasGenre ?genre), (?genre rdf:type
ont:Thriller) -> (?movie rdf:type ont:ThrillerMovie)]
[ruleGenreClassification1: (?movie ont:hasGenre ?genre), (?genre rdf:type
ont:Romance) -> (?movie rdf:type ont:RomanceMovie)]
```

- Jena6\_rules after modifications:

```
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.*;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.util.FileManager;
```

```
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.rdf.model.Statement;
import org.apache.jena.rdf.model StmtIterator;
import org.apache.jena.reasoner.Reasoner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

```

import java.util.Iterator;

public class Jena6_rules {

    public static void main(String[] args)

    {

        Model model = ModelFactory.createDefaultModel();

        model.read( "data/Movies1.owl" );

        Reasoner reasoner = new GenericRuleReasoner( Rule.rulesFromURL( "data/rul1.txt" ) );

        InfModel infModel = ModelFactory.createInfModel( reasoner, model );

        StmtIterator it = infModel.listStatements();

        while ( it.hasNext() )

        {

            Statement stmt = it.nextStatement();

            Resource subject = stmt.getSubject();

            Property predicate = stmt.getPredicate();

            RDFNode object = stmt.getObject();

            System.out.println( subject.toString() + " " + predicate.toString() + " " + object.toString()
);

        }

    }

}

```

- Output:

```

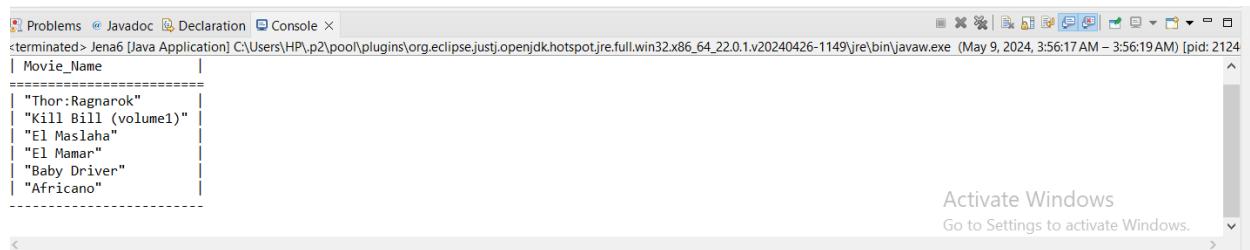
<terminated> Jena6_rules [Java Application] C:\Users\HP.p2\pool\plugins\org.eclipse.jst.openjdkhotspot\jre\full\win32x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (May 13, 2024, 4:56:42 AM, 4:56:44 AM) [pid: 28404]
/hp/ontologies/2024/3/Movies#There_Will_Be_Blood http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ThrillerMovie
/hp/ontologies/2024/3/Movies#Forrest_Gump http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/hp/ontologies/2024/3/Movies#RomanceMovie
/hp/ontologies/2024/3/Movies#Taika_Waititi http://www.semanticweb.org/hp/ontologies/2024/3/Movie#ActorAndDirector http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Jojo_Rabbit
/hp/ontologies/2024/3/Movies#Taika_Waititi http://www.semanticweb.org/hp/ontologies/2024/3/Movie#ActorAndDirector http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Thor_Ragnarok
/hp/ontologies/2024/3/Movies#Taika_Waititi http://www.semanticweb.org/hp/ontologies/2024/3/Movie#ActorAndWriter http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Jojo_Rabbit
/hp/ontologies/2024/3/Movies#El_Masla http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ThrillerMovie
/hp/ontologies/2024/3/Movies#Quentin_Tarantino http://www.semanticweb.org/hp/ontologies/2024/3/Movie#ActorAndWriter http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill
/hp/ontologies/2024/3/Movies#Quentin_Tarantino http://www.semanticweb.org/hp/ontologies/2024/3/Movie#ActorAndDirector http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction
/hp/ontologies/2024/3/Movies#Quentin_Tarantino http://www.semanticweb.org/hp/ontologies/2024/3/Movie#ActorAndDirector http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill
/hp/ontologies/2024/3/Movies#Quentin_Tarantino http://www.semanticweb.org/hp/ontologies/2024/3/Movie#ActorAndWriter http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction
/hp/ontologies/2024/3/Movies#Quentin_Tarantino http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ThrillerMovie
/hp/ontologies/2024/3/Movies#Uma_Thurman http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ActorAndWriter http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Kill_Bill
/hp/ontologies/2024/3/Movies#Uma_Thurman http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ActorAndWriter http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Pulp_Fiction

```

a. Movie with specific genre

```
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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT ?Movie_Name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasTitle ?Movie_Name.
?movie ont:hasGenre ?genre.
FILTER(?genre = "Action")
}
```



Movie_Name
"Thor:Ragnarok"
"Kill Bill (volume1)"
"El Maslaha"
"El Mamar"
"Baby Driver"
"Africano"

b. Actors with age bigger than 51

```
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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT ?actorName
```

```

WHERE {

?actor rdf:type ont:Actor.
?actor ont:hasAge ?age.
?actor ont:hasName ?actorName.
FILTER(?age > 51)
}

```

```

actorName
=====
| "John Travolta"
| "Tom Hanks"
| "Ahmed Ezz"
| "Quentin Tarantino"
=====

Activate Windows

```

- Jena6 code:

```

import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.util.FileManager;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
public class Jena6 {
    public static void main(String[] args) {
        // Load the ontology
        Model model = ModelFactory.createDefaultModel();
        FileManager fileManager = FileManager.get();
        String owlFile = "data/Movies.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);

        // Execute and display results for Rule 1
        executeRule(model, "data/age_rule.txt");

        // Execute and display results for Rule 2
        executeRule(model, "data/genre_rule.txt");
    }
}

```

```

private static void executeRule(Model model, String ruleFile) {
    // Read the SPARQL query from file
    String queryString = readQueryFromFile(ruleFile);

    // Execute the query
    try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
        ResultSet results = qexec.execSelect();

        // Print the results
        ResultSetFormatter.out(results);
    }
}

// Helper method to read the query from file
private static String readQueryFromFile(String queryFile) {
    StringBuilder sb = new StringBuilder();
    try (InputStream inputStream = FileManager.get().open(queryFile)) {
        if (inputStream != null) {
            String line;
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,
"UTF-8"));
            while ((line = reader.readLine()) != null) {
                sb.append(line).append("\n");
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return sb.toString();
}
}

```

The screenshot shows the Eclipse IDE's Console view displaying the results of a SPARQL query execution. The results are presented in two tables:

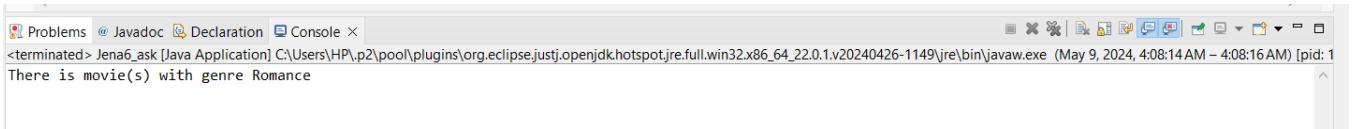
actorName
"John Travolta"
"Tom Hanks"
"Ahmed Ezz"
"Quentin Tarantino"

Movie_Name
"Thor:Ragnarok"
"Kill Bill (volume1)"
"El Maslaha"
"El Mamar"
"Baby Driver"
"Africano"

c. ASK query of specific genre (Romance)

```
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 ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
ASK
WHERE {
    ?movie rdf:type ont:Movies.
    ?movie ont:hasTitle ?title.
    ?movie ont:hasGenre ?genre.
    FILTER(?genre = "Romance") }
```



- Jena6\_ask code:

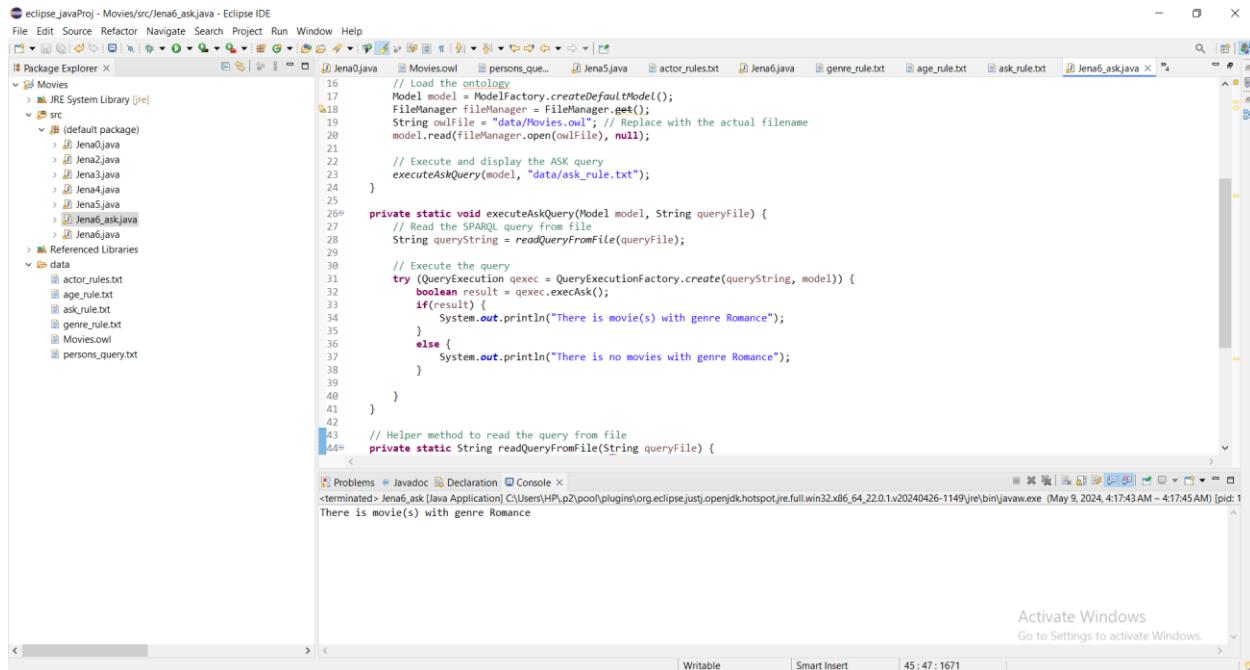
```
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.util.FileManager;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
public class Jena6_ask {
    public static void main(String[] args) {
        // Load the ontology
        Model model = ModelFactory.createDefaultModel();
        FileManager fileManager = FileManager.get();
        String owlFile = "data/Movies.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);
        // Execute and display the ASK query
        executeAskQuery(model, "data/ask_rule.txt");
    }
}
```

```
private static void executeAskQuery(Model model, String queryFile) {
    // Read the SPARQL query from file
    String queryString = readQueryFromFile(queryFile);

    // Execute the query
    try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
        boolean result = qexec.execAsk();
        if(result) {
            System.out.println("There is movie(s) with genre Romance");
        }
        else {
            System.out.println("There is no movies with genre Romance");
        }

    }
}

// Helper method to read the query from file
private static String readQueryFromFile(String queryFile) {
    StringBuilder sb = new StringBuilder();
    try (InputStream inputStream = FileManager.get().open(queryFile)) {
        if (inputStream != null) {
            String line;
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,
"UTF-8"));
            while ((line = reader.readLine()) != null) {
                sb.append(line).append("\n");
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return sb.toString();
}
}
```



- Jena6\_describe code:

```

import java.io.InputStream;
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.util.FileManager;

public class Jena6_describe {
    public static void main(String[] args) {
        // Load the ontology
        Model model = ModelFactory.createDefaultModel();
        FileManager fileManager = FileManager.get();
        String owlFile = "data/Movies.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);
    }
}

```

```
// Read the SPARQL query from file

String queryFile = "data/describe_actor.txt";
String queryString = readQueryFromFile(queryFile);

// Execute the DESCRIBE query

try (QueryExecution qexec = QueryExecutionFactory.create(QueryFactory.create(queryString),
model)) {

    Model describeModel = qexec.execDescribe();

    describeModel.write(System.out, "TURTLE"); // Print the result in Turtle format

}

// Helper method to read the query from file

private static String readQueryFromFile(String queryFile) {

    StringBuilder sb = new StringBuilder();

    try (InputStream inputStream = FileManager.get().open(queryFile)) {

        if (inputStream != null) {

            String line;

            java.io.BufferedReader reader = new java.io.BufferedReader(
                new java.io.InputStreamReader(inputStream, "UTF-8"));

            while ((line = reader.readLine()) != null) {

                sb.append(line).append("\n");

            }

        }

    } catch (java.io.IOException e) {

        e.printStackTrace();

    }

    return sb.toString();

}

}
```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows files like `Movies.owl`, `Jena5.java`, `actor_rules.txt`, `Jena6.java`, `genre_rule.txt`, `age_rule.txt`, `Jena6_ask.java`, `Jena1.java`, `Jena6_describe.java`, and `describe_actor.txt`.
- Code Editor:** Displays the Java code for `Jena6_describe.java`. The code imports Java IO, Jena RDF Model, and Util classes. It reads a SPARQL query from a file, creates a Model, and executes a DESCRIBE query to print results in Turtle format.
- Console:** Shows the output of the Java application, which includes prefixes and a SPARQL query result.
- Status Bar:** Shows "Writable", "Smart Insert", and the date/time "11:46:405".

- **describe\_actor.txt:**

DESCRIBE <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Actor>