MOVIES ONTOLOGY

CSE488 – Ontologies and Semantic Web

Team 2

REDA MOHSEN REDA GEBRIL

ID: 18P5141

NOOR-ELDIN TALAAT EZZAT

ID: 18P3826

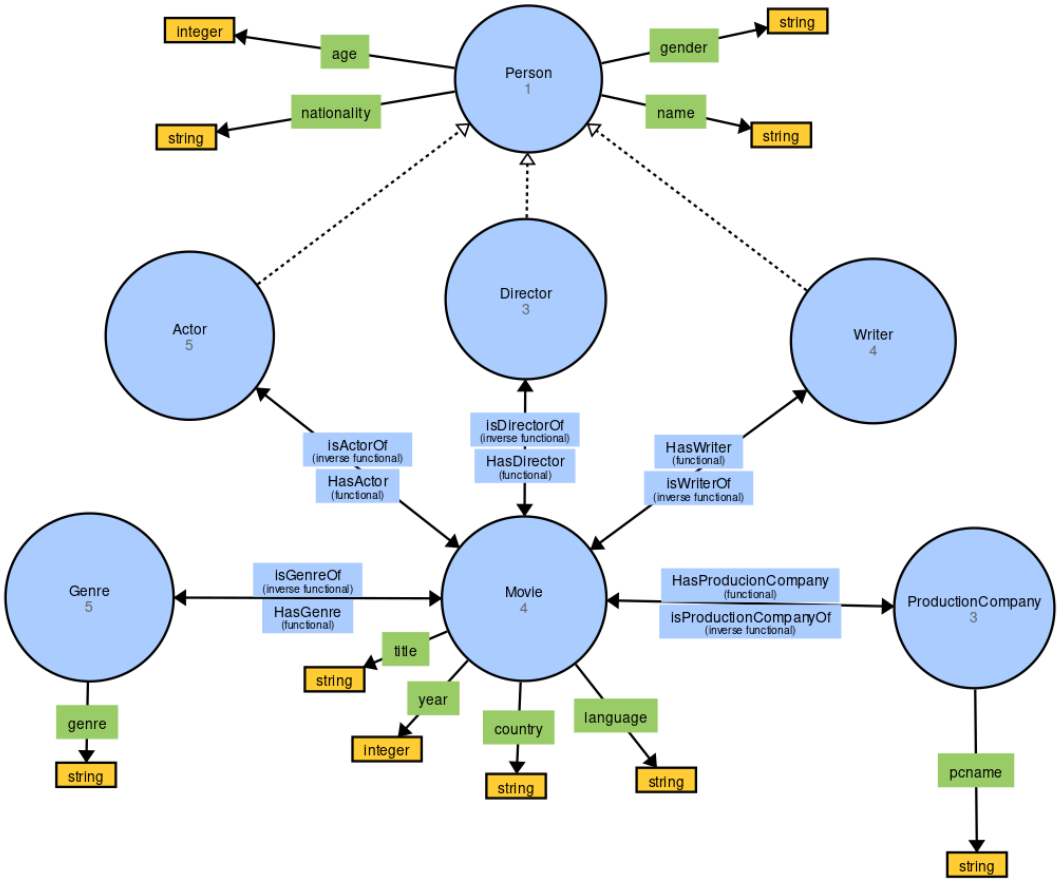
ZEYAD YASSER ABDALLAH ALI

ID: 18P4353

Github Link

reda-mohsen/Movies_Ontology: CSE488 Ontologies and the Semantic Web Project (github.com)

Part I Modeling the Ontology



Classes

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

Data properties

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

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

5. pcname: domain = ProductionCompany, range = xsd:string, characteristic = functional.

   Indicates the name of a production company.
6. country: domain = Movie, range = xsd:string, characteristic = functional.
7. language: domain = Movie, range = xsd:string, characteristic = functional.
8. title: domain = Movie, range = xsd:string, characteristic = functional.
9. year: domain = Movie, range = xsd:integer, characteristic = functional.
10. genre: domain = Genre, range = xsd:string, characteristic = functional.

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

Object Properties

1. HasActor: domain = Movie, range = Actor, characteristic = functional

   Indicates that a movie has an actor.

2. HasDirector: domain = Movie, range = Director, characteristic = functional

   Indicates that a movie has a director

3. HasWriter: domain = Movie, range = Writer, characteristic = functional

   Indicates that a movie has a writer.

4. isActorOf: domain = Actor, range = Movie, characteristic = inverse functional

   Indicates that an actor has acted in a movie.

5. isDirectorOf: domain = Director, range = Movie, characteristic = functional

   Indicates that a movie is directed by a director.

6. isWriterOf: domain = Writer, range = Movie, characteristic = functional

   Indicates that a movie is written by a writer

7. HasGenre: domain = Movie, range = Genre, characteristic = functional

   Indicates that a movie has a genre.

8. isGenreOf: domain = Genre, range = Movie, characteristic = inverse functional

   Indicates that a genre belongs to a movie.

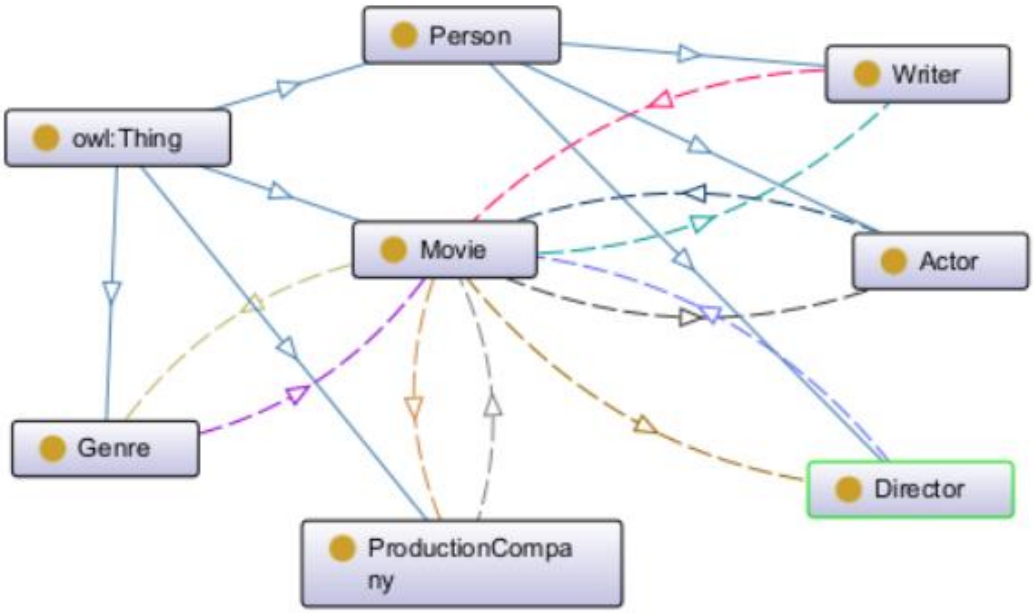
9. HasProductionCompany; domain = Movie, Range = ProductionCompany, Characteristic = functional.
10. isProductionCompanyOf; domain = ProductionCompany, Range = Movie, Characteristic = inverse functional.

    Indicates that a production company produces a movie.

Constraints

1. Class Movie, Genre, ProductionCompany, and Person are disjoint classes.

2. Class Actor, Movie, ProductionCompany, and Genre are disjoint classes.
3. Class Director, Movie, ProductionCompany, and Genre are disjoint classes.
4. Class Writer, Movie, ProductionCompany, and Genre are disjoint classes.
5. A movie must have at least one actor (min 1 HasActor).
6. A movie must have exactly one director (exact 1 HasDirector).
7. A movie must have at least one writer (min 1 HasWriter).
8. An actor must have acted in at least one movie (min 1 isActorOf).
9. A director must have directed at least one movie (min 1 isDirectorOf).
10. A writer must have written for at least one movie (min 1 isWriterOf).
11. A movie must have at least one genre (min 1 genre).
12. A genre must have at least one movie (min 1 movie)
13. A movie must be produced by at least one production company (min 1 HasProductionCompany).
14. A production company must have produced at least one movie(min 1 Movie).
15. year some xsd:integer [<= 2023]
16. (gender value "female") or (gender value "male")
17. age some xsd:integer [> 0]



Part II Populating Ontology with Individuals

Individuals

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

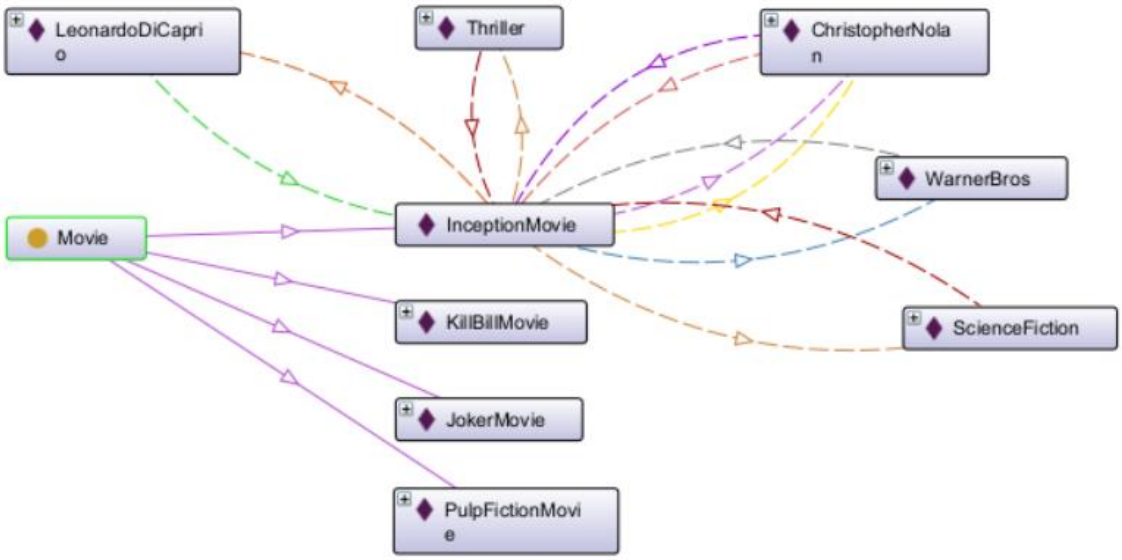
2. Some individuals to the Movie class:
   a) Pulp Fiction, Genre: Crime Thriller, 1994, USA, English, Production Company: Miramax Films and A Band Apart.
   b) Kill Bill (volume 1), Genre: Action Crime Thriller, 2003, USA, English, produced by A Band Apart.
   c) Inception, Genre = Thriller, Science Fiction, 2010, USA, English, Produced by Warner Bros.
   d) Joker, Genre = Crime Drama, 2019, USA, English, produced by Warner Bros.

3. Some individuals to the Person class:
   a) Quentin Tarantino, American, 53 years old, writer and director of Pulp Fiction and Kill Bill (volume1). He also played a role in that movie.
   b) John Travolta, American, 59 years old, actor in Pulp Fiction.
   c) Uma Thurman, American, 43 years old, actress in Pulp Fiction. She also participated as a writer in Kill Bill (volume1).
   d) Christopher Nolan, British-American, 51 years old, writer and director of Inception.
   e) Leonardo DiCaprio, American, 47 years old, actor in Inception.
   f) Joaquin Phoenix, American, 47 years old, actor in Joker.
   g) Todd Phillips, American, 51 years old, writer and director of Joker.

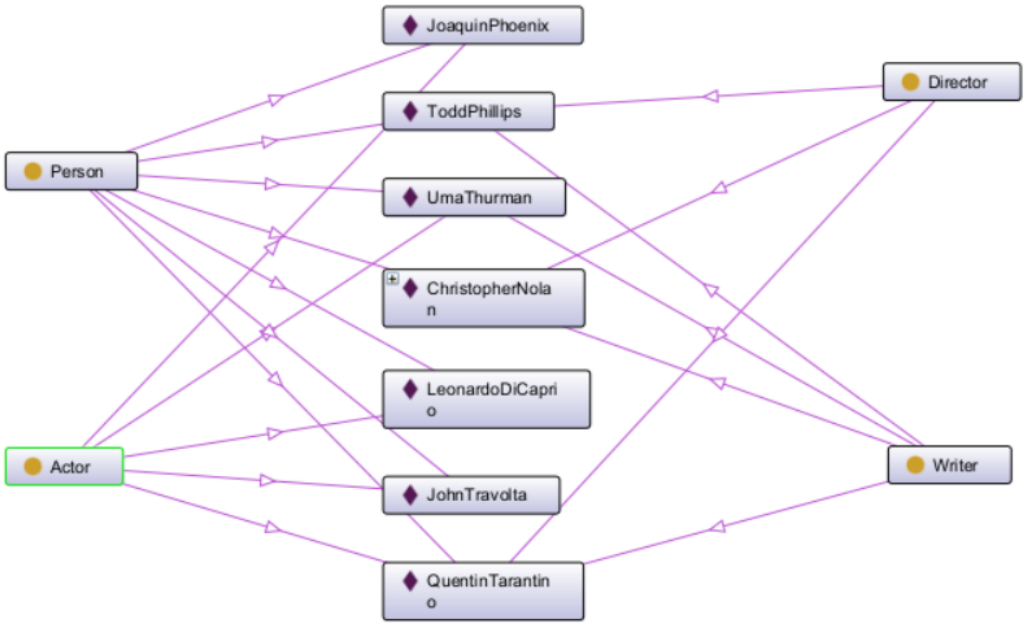
4. Some individuals to the ProductionCompany class:
   a) Miramax Films, a production company of Pulp Fiction.
   b) A Band Apart, the production company of Pulp Fiction and Kill Bill.
   c) Warner Bros, the production company of Joker and Inception.

Ontology Graph

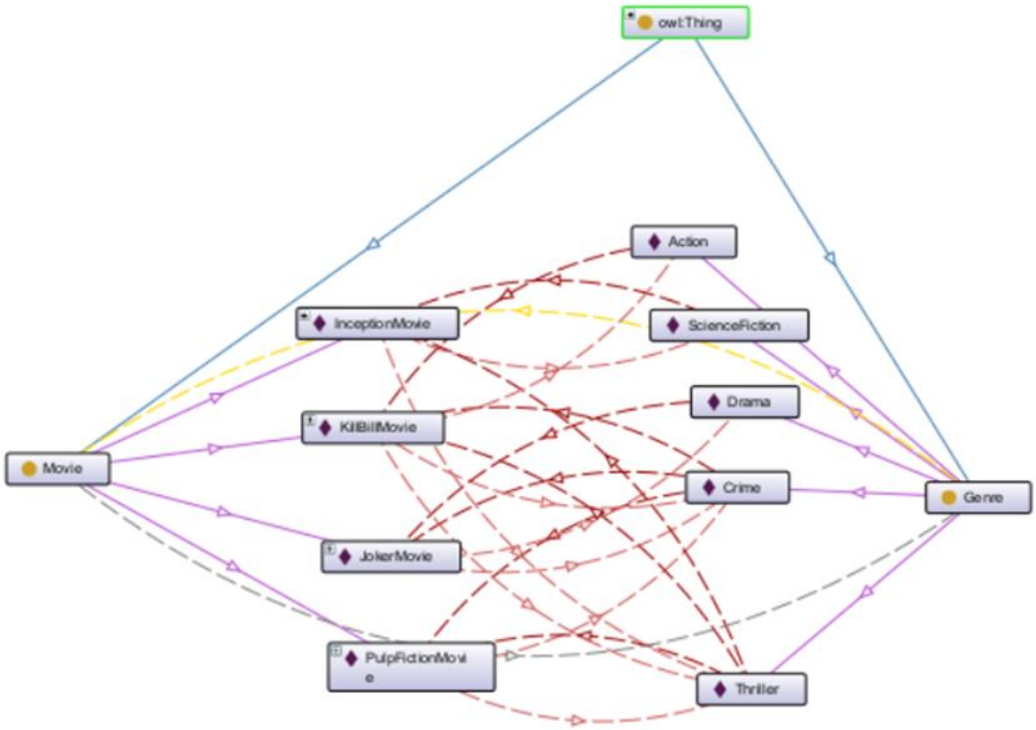
Movie Instances



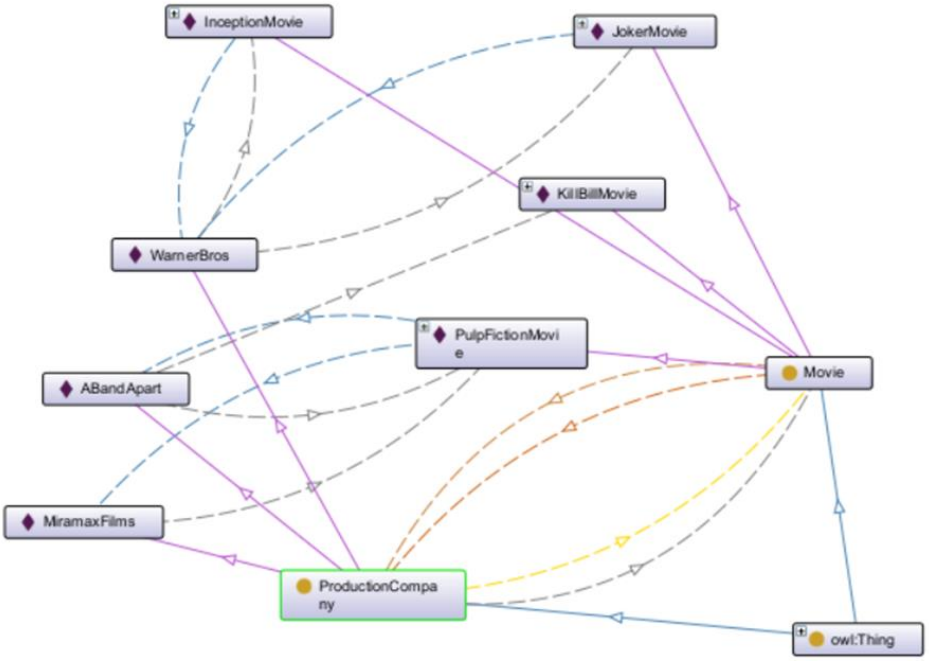
Person Instances



Genre Instances



Production Company Instances



Part III Sparql Queries

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

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

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

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

PREFIX ont: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>

SELECT DISTINCT ?name ?age ?nationality

WHERE {

```
    ?actor rdf:type ont:Actor.

    ?actor ont:name ?name.

    OPTIONAL {

        ?actor ont:age ?age

  }

    OPTIONAL {

        ?actor ont:nationality ?nationality

  }
```

}

Execute

?name	?age	?nationality
Joaquin Phoenix	47	
Quentin Tarantino	53	American
Quentin Tarantino	43	American
Uma Thurman	53	American
Uma Thurman	43	American
Leonardo DiCaprio		American
Quentin Tarantino	59	American
John Travolta	53	American
John Travolta	59	American
John Travolta	43	American
Uma Thurman	59	American

2. In this example, 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 owl: <http://www.w3.org/2002/07/owl#>

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

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

PREFIX ont: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>

SELECT  ?title ?year ?genre_name ?actor_name ?director_name

WHERE {

?movie rdf:type ont:Movie.

?movie ont:title ?title.

?movie ont:year ?year.

?movie ont:HasGenre ?genre.

?genre ont:genre ?genre_name.

{

?movie ont:HasActor ?actor_name.

}

UNION

{

?movie ont:HasDirector ?director_name.

}

FILTER(?year < 2010 && ?genre_name = "Action")

}
```

Execute

?title	?year	?genre_name	?actor_name	?director_name
Kill Bill (volume 1)	2003	Action	ont:UmaThurman	
Kill Bill (volume 1)	2003	Action	ont:JohnTravolta	
Kill Bill (volume 1)	2003	Action	ont:QuentinTarantino	
Kill Bill (volume 1)	2003	Action		ont:UmaThurman
Kill Bill (volume 1)	2003	Action		ont:JohnTravolta
Kill Bill (volume 1)	2003	Action		ont:QuentinTarantino
Pulp Fiction	1994	Action	ont:UmaThurman	
Pulp Fiction	1994	Action	ont:JohnTravolta	
Pulp Fiction	1994	Action	ont:QuentinTarantino	
Pulp Fiction	1994	Action		ont:UmaThurman
Pulp Fiction	1994	Action		ont:JohnTravolta
Pulp Fiction	1994	Action		ont:QuentinTarantino

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

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movies: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>

CONSTRUCT {
    ?person rdf:type movies:ActorDirector .
}
WHERE {
    ?person rdf:type movies:Actor .
    ?person rdf:type movies:Director .
}
```

```java

        // Read the SPARQL query from file
        String queryFile = "data/movie_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 persons who are both actors and directors
        System.out.println("Persons who are both actors and directors:");
        resultModel.listResourcesWithProperty(resultModel.createProperty("http://www.w3.org/1999/02/22-rdf-syntax-ns#type"),
                resultModel.createResource("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/ActorDirector"))
                .forEachRemaining(resource -> System.out.println(resource.getURI().toString()));
    }
```

```java
        // Read the SPARQL query from file
        String queryFile = "data/movie_rules.txt";
        String queryString = readQueryFromFile(queryFile);

```

Outline Problems Console ×

```
<terminated> Jena5 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (May 19, 2023, 10:01:17 PM – 10:01:19 PM) [pid: 14096]
Persons who are both actors and directors:
http://www.semanticweb.org/redar/ontologies/2023/4/Movies/QuentinTarantino
```

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

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

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

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>
ASK
WHERE {
  ?movie rdf:type ont:Movie .
  ?movie ont:title "Inception" .
}
```

```java

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

        // Execute the ASK query
        try (
            QueryExecution qexec = QueryExecutionFactory.create(QueryFactory.create(queryString), model)) {
            boolean movieExists = qexec.execAsk();
            if (movieExists) {
                System.out.println("The movie 'Inception' exists in the ontology.");
            } else {
                System.out.println("The movie 'Inception' does not exist in the ontology.");
            }
```

Outline Problems Console ×

```
<terminated> Jena6 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (May 19, 2023, 10:02:27 PM – 10:02:30 PM) [pid: 18384]
The movie 'Inception' exists in the ontology.
```

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

DESCRIBE <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/Actor>

```java
        // Read the SPARQL query from file
        String queryFile = "data/ask.txt";
        String queryString = readQueryFromFile(queryFile);

        // Execute the ASK query
        try (
            QueryExecution qexec = QueryExecutionFactory.create(QueryFactory.create(queryString), model)) {
            boolean movieExists = qexec.execAsk();
            if (movieExists) {
                System.out.println("The movie 'Inception' exists in the ontology.");
            } else {
                System.out.println("The movie 'Inception' does not exist in the ontology.");
            }
        }
    }
```

```java

        // 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
        }
    }
```

Outline Problems Console ×

```
<terminated> Jena7 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (May 19, 2023, 10:03:00 PM – 10:03:02 PM) [pid: 19644]
@prefix :                    <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/> .
@prefix owl:                 <http://www.w3.org/2002/07/owl#> .
@prefix rdf:                 <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:                <http://www.w3.org/2000/01/rdf-schema#> .
@prefix untitled-ontology-3: <http://www.semanticweb.org/redar/ontologies/2023/4/untitled-ontology-3#> .
@prefix xsd:                 <http://www.w3.org/2001/XMLSchema#> .

:Actor  rdf:type         owl:Class ;
        rdfs:comment     "Individuals who play a role in a movie." ;
        rdfs:subClassOf  :Person ;
        rdfs:subClassOf  [ rdf:type                     owl:Restriction ;
                           owl:minQualifiedCardinality  "1"^^xsd:nonNegativeInteger ;
                           owl:onClass                  :Movie ;
                           owl:onProperty               :isActorOf
                         ] .
```

6. List all persons in the ontology showing if they are actor, director, and writer.

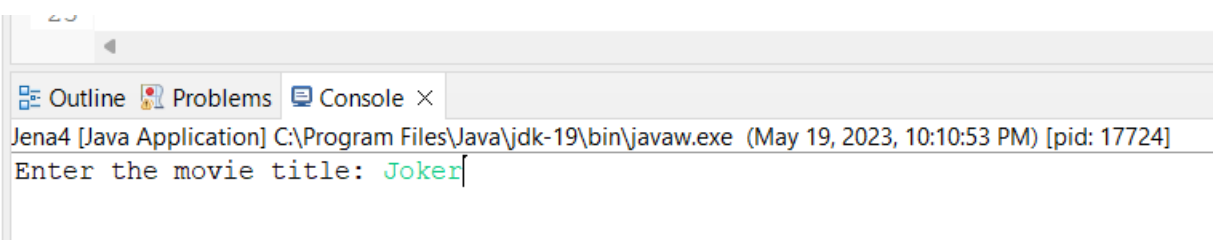
```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX movies: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>

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

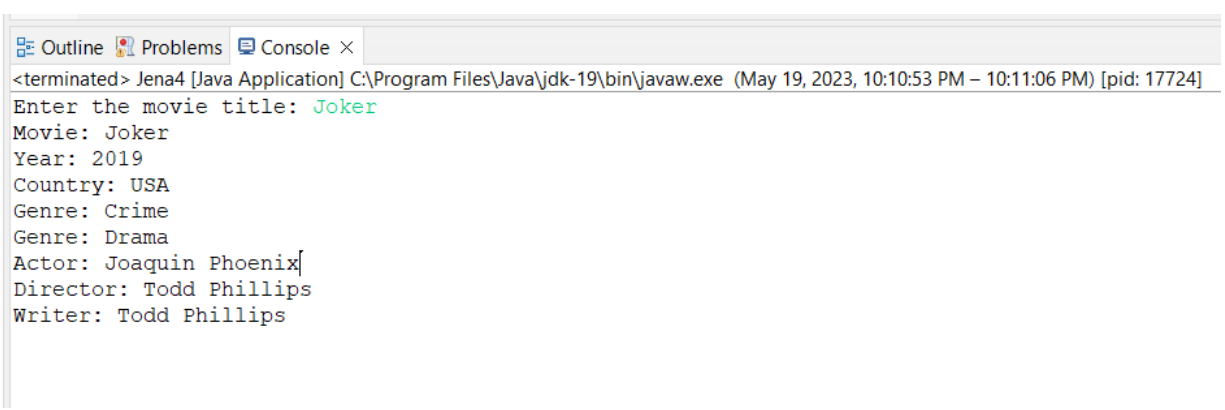
```
Outline  Problems  Console ×
<terminated> Jena1 [Java Application] C:\Program Files\Java\j
Person: Uma Thurman
 - Actor
 - Writer
Person: Joaquin Phoenix
 - Actor
Person: Quentin Tarantino
 - Actor
 - Director
 - Writer
Person: Leonardo DiCaprio
 - Actor
Person: Todd Phillips
 - Director
 - Writer
Person: John Travolta
 - Actor
Person: Christopher Nolan
 - Director
 - Writer
```

7. List the movie titles and its details.

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX movies: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>

SELECT DISTINCT ?title ?year ?country ?genre ?actor ?director ?writer
?company
WHERE {
  ?movie rdf:type movies:Movie ;
         movies:title ?title ;
         movies:year ?year ;
         movies:country ?country ;
         movies:HasGenre ?genreResource ;
         movies:HasActor ?actorResource ;
         movies:HasDirector ?directorResource ;
         movies:HasWriter ?writerResource ;
         movies:HasProductionCompany ?productionCompany.


  ?genreResource movies:genre ?genre .

  ?actorResource movies:name ?actor .

  ?directorResource movies:name ?director .

  ?writerResource movies:name ?writer .

  ?productionCompany movies:pcname ?company.
   }
```

But in this example, we make the user enter the name of the movie and if found in the ontology, it displays its details.





Part IV Manipulating the Ontology using Jena

1. In this example, we manipulate the ontology using Jena library.

Jena0.java × | Jena1.java | persons_que... | Jena3.java | Jena4.java | movies_quer... | Jena5.java | movie_rules.txt | Jena6.java | ask.txt | describe.txt

```java
package MoviePack;

import org.apache.jena.ontology.*;

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";
        model.read(fileManager.open(owlFile), null);
        // List all movie titles
        OntClass movieClass = model.getOntClass("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/Movie");
        ExtendedIterator<? extends OntResource> movieIterator = movieClass.listInstances();
        while (movieIterator.hasNext()) {
            OntResource movieResource = movieIterator.next();
            // Get the title property of the movie
            Property titleProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/title");
            // Get the value of the title property
            RDFNode titleNode = movieResource.getPropertyValue(titleProperty);
            if (titleNode != null && titleNode.isLiteral()) {
                // Extract the title value as a string
                String title = titleNode.asLiteral().getString();
                // Print the movie title
                System.out.println("Movie Title: " + title);
            }
        }
    }
}
```

2. Displays all the Persons (without using queries, without inference), showing if they are an actor, writer, and director.

```java
public class Jena1 {
    public static void main(String[] args) {
        // Load the ontology
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        FileManager fileManager = FileManager.get();
        String owlFile = "data/Movies.owl";
        model.read(fileManager.open(owlFile), null);

        // List all persons
        OntClass personClass = model.getOntClass("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/Person");
        ExtendedIterator<? extends OntResource> personIterator = personClass.listInstances();
        while (personIterator.hasNext()) {
            OntResource personResource = personIterator.next();

            // Retrieve the name of the person
            Property nameProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/name");
            StmtIterator personNameIterator = personResource.listProperties(nameProperty);
            while (personNameIterator.hasNext()) {
                Statement personNameStatement = personNameIterator.next();
                RDFNode personNameNode = personNameStatement.getObject();
                if (personNameNode != null && personNameNode.isLiteral()) {
                    String personName = personNameNode.asLiteral().getString();
                    System.out.println("Person: " + personName);
                }
            }
```

```java
            }

            // Check if the person is also an actor, director, or writer
            if (personResource.hasRDFType(model.getOntClass("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/Actor"))) {
                System.out.println(" - Actor");
            }
            if (personResource.hasRDFType(model.getOntClass("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/Director"))) {
                System.out.println(" - Director");
            }
            if (personResource.hasRDFType(model.getOntClass("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/Writer"))) {
                System.out.println(" - Writer");
            }
        }
    }
}
```

Outline Problems Console ×

```
<terminated> Jena1 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (May 19, 2023, 10:16:44 PM –
Person: Uma Thurman
 - Actor
 - Writer
Person: Joaquin Phoenix
 - Actor
Person: Quentin Tarantino
 - Actor
 - Director
 - Writer
Person: Leonardo DiCaprio
 - Actor
Person: Todd Phillips
 - Director
 - Writer
Person: John Travolta
 - Actor
Person: Christopher Nolan
 - Director
 - Writer
```

3. Displays all the actors ((without using queries, using inference))

```java
public class Jena3 {
    public static void main(String[] args) {
        // Load the ontology
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        FileManager fileManager = FileManager.get();
        String owlFile = "data/Movies.owl";
        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/redar/ontologies/2023/4/Movies/Actor");
        ExtendedIterator<? extends OntResource> actorIterator = actorClass.listInstances();
        while (actorIterator.hasNext()) {
            OntResource actorResource = actorIterator.next();
            Property nameProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/name");
            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);
                }
            }
        }
    }
}
```

```
Outline  Problems  Console ×
<terminated> Jena3 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (May 19, 2023, 10:17:48 PM – 10:17:50 PM) [pid: 21732]
Actor: Uma Thurman
Actor: Quentin Tarantino
Actor: Leonardo DiCaprio
Actor: John Travolta
Actor: Joaquin Phoenix
```

4. Taking from user a movie title, and if found in the ontology print its details.

```java
public class Jena4 {
    public static void main(String[] args) {
        // Load the ontology
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        FileManager fileManager = FileManager.get();
        String owlFile = "data/Movies.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);

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

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

        // Find the movie with the given title
        OntClass movieClass = model.getOntClass("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/Movie");
        ExtendedIterator<? extends OntResource> movieIterator = movieClass.listInstances();
        OntResource targetMovie = null;
        while (movieIterator.hasNext()) {
            OntResource movieResource = movieIterator.next();
            Property titleProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/title");
            RDFNode titleNode = movieResource.getPropertyValue(titleProperty);
            if (titleNode != null && titleNode.isLiteral() && titleNode.asLiteral().getString().equals(movieTitle)) {
                targetMovie = movieResource;
                break;
            }
        }


        // Check if the movie exists
        if (targetMovie == null) {
            System.out.println("Error: Movie not found");
        } else {
            // Display the Movie Title
            System.out.println("Movie: " + movieTitle);

            ////////////////////////////////
            // Display the movie details //
            ////////////////////////////////

            // Display the Year of Release
            Property yearProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/year");
            RDFNode yearNode = targetMovie.getPropertyValue(yearProperty);
            if (yearNode != null && yearNode.isLiteral()) {
                int year = yearNode.asLiteral().getInt();
                System.out.println("Year: " + year);
            }

            // Display the Country
            Property countryProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/country");
            StmtIterator countryIterator = targetMovie.listProperties(countryProperty);
            while (countryIterator.hasNext()) {
                Statement countryStatement = countryIterator.next();
                RDFNode countryNode = countryStatement.getObject();
                if (countryNode != null && countryNode.isLiteral()) {
                    String country = countryNode.asLiteral().getString();
                    System.out.println("Country: " + country);
                }
            }

```

```java
            // Display the Genre
            Property genreProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/HasGenre");
            StmtIterator genreIterator = targetMovie.listProperties(genreProperty);
            while (genreIterator.hasNext()) {
                Statement genreStatement = genreIterator.next();
                RDFNode genreNode = genreStatement.getObject();
                if (genreNode != null && genreNode.isResource()) {
                    Resource genreResource = genreNode.asResource();
                    Property genreNameProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/genre");
                    StmtIterator genreGenreIterator = genreResource.listProperties(genreNameProperty);
                    while (genreGenreIterator.hasNext()) {
                        Statement genreGenreStatement = genreGenreIterator.next();
                        RDFNode genreGenreNode = genreGenreStatement.getObject();
                        if (genreGenreNode != null && genreGenreNode.isLiteral()) {
                            String genre = genreGenreNode.asLiteral().getString();
                            System.out.println("Genre: " + genre);
                        }
                    }
                }
            }
```

```java
            // Display the Actor
            Property actorProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/HasActor");
            StmtIterator actorIterator = targetMovie.listProperties(actorProperty);
            while (actorIterator.hasNext()) {
                Statement actorStatement = actorIterator.next();
                RDFNode actorNode = actorStatement.getObject();
                if (actorNode != null && actorNode.isResource()) {
                    Resource actorResource = actorNode.asResource();
                    Property nameProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/name");
                    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);
                        }
                    }
                }
            }
```

```java
            // Display the Director
            Property directorProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/HasDirector");
            StmtIterator directorIterator = targetMovie.listProperties(directorProperty);
            while (directorIterator.hasNext()) {
                Statement directorStatement = directorIterator.next();
                RDFNode directorNode = directorStatement.getObject();
                if (directorNode != null && directorNode.isResource()) {
                    Resource directorResource = directorNode.asResource();
                    Property nameProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/name");
                    StmtIterator directorNameIterator = directorResource.listProperties(nameProperty);
                    while (directorNameIterator.hasNext()) {
                        Statement directorNameStatement = directorNameIterator.next();
                        RDFNode directorNameNode = directorNameStatement.getObject();
                        if (directorNameNode != null && directorNameNode.isLiteral()) {
                            String directorName = directorNameNode.asLiteral().getString();
                            System.out.println("Director: " + directorName);
                        }
                    }
                }
            }
```

```java
            // Display the Writer
            Property writerProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/HasWriter");
            StmtIterator writerIterator = targetMovie.listProperties(writerProperty);
            while (writerIterator.hasNext()) {
                Statement writerStatement = writerIterator.next();
                RDFNode writerNode = writerStatement.getObject();
                if (writerNode != null && writerNode.isResource()) {
                    Resource writerResource = writerNode.asResource();
                    Property nameProperty = model.getProperty("http://www.semanticweb.org/redar/ontologies/2023/4/Movies/name");
                    StmtIterator writerNameIterator = writerResource.listProperties(nameProperty);
                    while (writerNameIterator.hasNext()) {
                        Statement writerNameStatement = writerNameIterator.next();
                        RDFNode writerNameNode = writerNameStatement.getObject();
                        if (writerNameNode != null && writerNameNode.isLiteral()) {
                            String writerName = writerNameNode.asLiteral().getString();
                            System.out.println("Writer: " + writerName);
                        }
                    }
                }
            }
        }
    }
}
```

```
Outline  Problems  Console ×
<terminated> Jena4 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (May 19, 2023, 10:24:01 PM – 10:24:08 PM) [pid: 11368]
Enter the movie title: Inception
Movie: Inception
Year: 2010
Country: USA
Genre: Science Fiction
Genre: Thriller
Actor: Leonardo DiCaprio
Director: Christopher Nolan
Writer: Christopher Nolan
```

Part V

Queries

```
String queryString = "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
    "PREFIX ont: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>\n" +
    "SELECT ?title ?year ?cont ?lan ?director_name\n" +
    "WHERE {\n" +
    "  ?movie rdf:type ont:Movie.\n" +
    "  ?movie ont:title ?title.\n" +
    "  ?movie ont:year ?year.\n" +
    "  ?movie ont:country ?cont.\n" +
    "  ?movie ont:language ?lan.\n" +

      "  ?movie ont:HasGenre ?genre.\n" +
      "  ?genre ont:genre ?genre_name.\n" +
    "  FILTER(?title = \""+NameList.getSelectedItem()+"\")\n" +
    "}";
```

```
queryString = "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
        "PREFIX ont: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>\n" +
        "SELECT ?title ?genre_name\n" +
        "WHERE {\n" +
        "  ?movie rdf:type ont:Movie.\n" +
        "  ?movie ont:title ?title.\n" +
        "  ?movie ont:HasGenre ?genre.\n" +
        "  ?genre ont:genre ?genre_name.\n" +
        "  FILTER(?title = \""+NameList.getSelectedItem()+"\")\n" +
        "}";
```

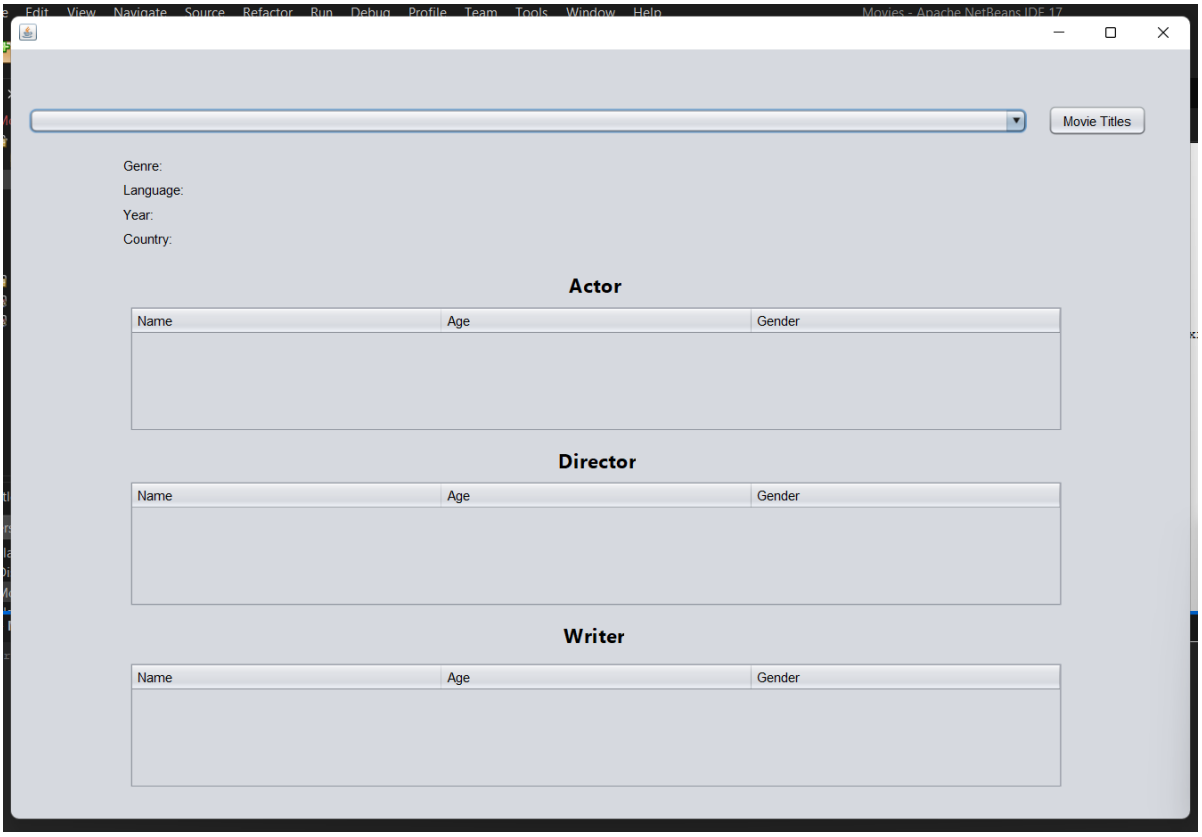
```
ArrayList<Person> director = new ArrayList<Person>();
queryString = "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
    "PREFIX ont: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>\n" +
    "SELECT ?title ?dir_name ?dir_age ?dir_gen \n" +
    "WHERE {\n" +
    "  ?movie rdf:type ont:Movie.\n" +
    "  ?movie ont:title ?title.\n" +
    "  ?movie ont:HasDirector ?dir.\n" +
    "  ?dir ont:name ?dir_name.\n" +
    "  ?dir ont:age ?dir_age.\n" +
    "  ?dir ont:gender ?dir_gen.\n" +
    "  FILTER(?title = \""+NameList.getSelectedItem()+"\")\n" +
    "}";
```

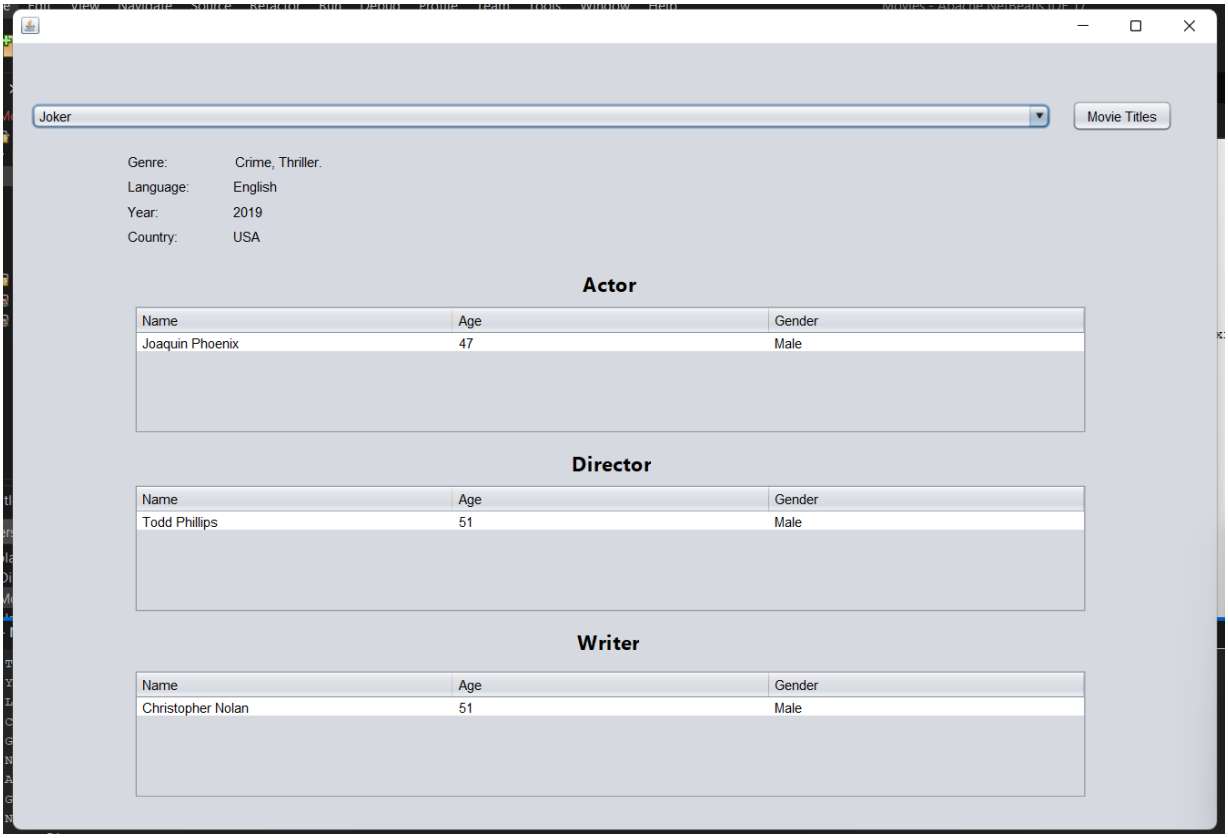
```
ArrayList<Person> actor = new ArrayList<Person>();
queryString = "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
    "PREFIX ont: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>\n" +
    "SELECT ?title ?actor_name ?actor_age ?actor_gen \n" +
    "WHERE {\n" +
    "  ?movie rdf:type ont:Movie.\n" +
    "  ?movie ont:title ?title.\n" +
    "  ?movie ont:HasActor ?actor.\n" +
    "  ?actor ont:name ?actor_name.\n" +
    "  ?actor ont:age ?actor_age.\n" +
    "  ?actor ont:gender ?actor_gen.\n" +
    "  FILTER(?title = \""+NameList.getSelectedItem()+"\")\n" +

    "}";
```

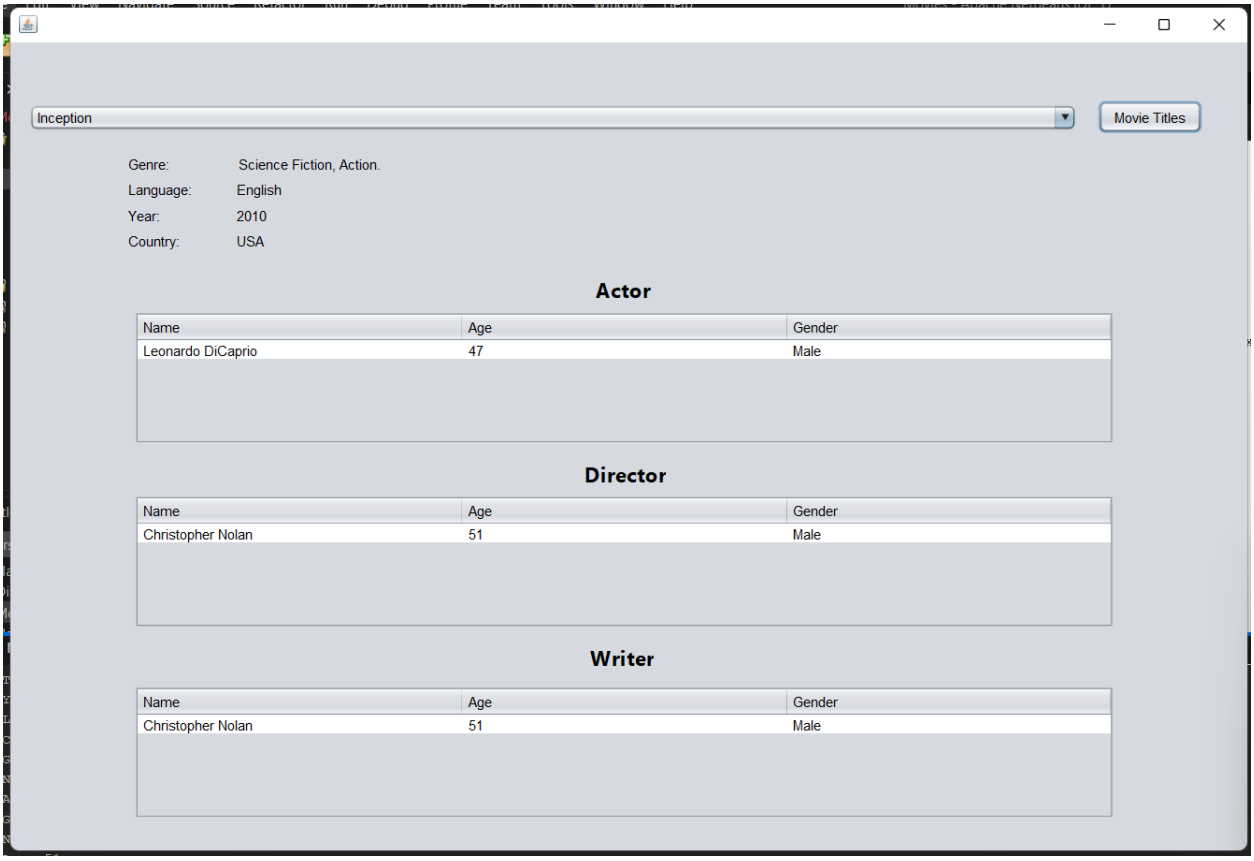
```
ArrayList<Person> writer = new ArrayList<Person>();
queryString = "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
    "PREFIX ont: <http://www.semanticweb.org/redar/ontologies/2023/4/Movies/>\n" +
    "SELECT ?title ?wir_name ?wir_age ?wir_gen \n" +
    "WHERE {\n" +
    "  ?movie rdf:type ont:Movie.\n" +
    "  ?movie ont:title ?title.\n" +
    "  ?movie ont:HasWriter ?wri.\n" +
    "  ?wir ont:name ?wir_name.\n" +
    "  ?wir ont:age ?wir_age.\n" +
    "  ?wir ont:gender ?wir_gen.\n" +
    "  FILTER(?title = \""+NameList.getSelectedItem()+"\")\n" +
    "}";
```

```
queryString = "PREFIX movies:<http://www.semanticweb.org/redar/ontologies/2023/4/Movies/> "
        + "SELECT  (str(?x) as ?Movie) "
         + "where { ?y movies:title ?x."
        +" }\n ORDER BY DESC(?title)";
```

GUI




Joker
Movie Titles
Genre: Crime, Thriller.
Language: English
Year: 2019
Country: USA
Actor
Name Age Gender
Joaquin Phoenix 47 Male
Director
Name Age Gender
Todd Phillips 51 Male
Writer
Name Age Gender
Christopher Nolan 51 Male


Inception
Movie Titles
Genre: Science Fiction, Action.
Language: English
Year: 2010
Country: USA
Actor
Name Age Gender
Leonardo DiCaprio 47 Male
Director
Name Age Gender
Christopher Nolan 51 Male
Writer
Name Age Gender
Christopher Nolan 51 Male