A picture containing logo

Description automatically generated

**AIN SHAMS UNIVERSITY**

**FACULTY OF ENGINEERING**

**AIN SHAMS UNIVERSITY**

**FACULTY OF ENGINEERING**

**CREDIT HOURS ENG. PROGRAM**

**Computer Engineering and Software Systems systems**

CSE488: Ontologies and the Semantic Web

Team 10

Project Report

Anass Zikry 20P6382

Ahmed Khaled 20P7387

Kerolos Noshy Kamel 21P0132

Youssef Mahmoud Ahmed 20p6649

Contents

[Problem Description 2](#_Toc166538596)

[Domain Entities: 2](#_Toc166538597)

[Genre Classification: 3](#_Toc166538598)

[Relationships: 3](#_Toc166538599)

[Requirements: 3](#_Toc166538600)

[Example: 3](#_Toc166538601)

[Data Flow Diagrams 4](#_Toc166538602)

[Context level 4](#_Toc166538603)

[Level 1 4](#_Toc166538604)

[Querying the ontology 5](#_Toc166538605)

[Ontology Visualization 22](#_Toc166538606)

[GitHub link 22](#_Toc166538607)

[Part 5 application 23](#_Toc166538608)

[Table of figures 30](#_Toc166538609)

# Problem Description

The goal is to create an ontology using the Protégé editor that models the domain of movies. The ontology will capture various aspects of movies including their titles, directors, writers, actors, genres, release year, country of production, and language. Additionally, it will represent individuals involved in the movie-making process such as directors, writers, and actors, each with their own attributes.

## Domain Entities:

1. **Movie:**
   * Attributes:
     + Title: The title of the movie.
     + Year: The year of release.
     + Country: The country of production.
     + Language: The language of the movie.
   * Relationships:
     + Directors: One or more directors who worked on the movie.
     + Writers: One or more writers who contributed to the screenplay.
     + Actors: One or more actors who appeared in the movie.
     + Genres: One or more genres that classify the movie.
2. **Person:**
   * Attributes:
     + Name: The name of the person.
     + Gender: The gender of the person (male or female).
     + Age: The age of the person.
     + Nationality: The nationality of the person.
3. **Genre:**
   * Attributes:
     + genre: The name of the genre.

## Genre Classification:

The genre of a movie can be classified into the following categories:

* Thriller
* Crime
* Action
* Drama
* Comedy
* Horror

## Relationships:

* **Directors, Writers, Actors:** Each movie can have one or more directors, writers, and actors associated with it. These individuals are considered persons in the ontology.
* **Genres:** Each movie can have one or more genres from the predefined list.

## Requirements:

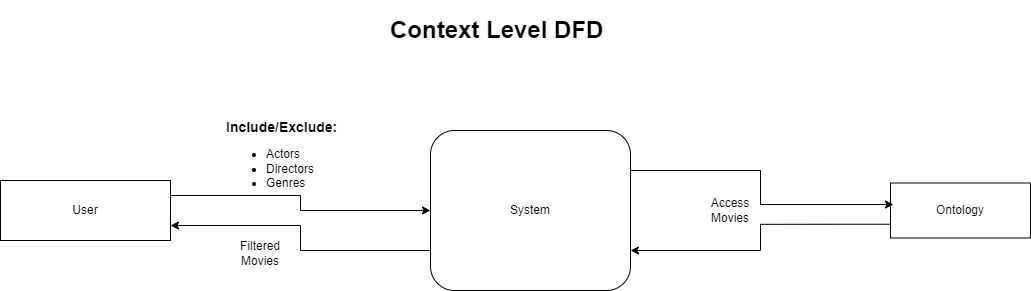
1. Create classes for Movie and Person.
2. Define properties for each attribute of Movie and Person classes.
3. Establish relationships between movies and individuals (directors, writers, actors).
4. Classify movies into genres using appropriate properties and classes.
5. Ensure that individuals (directors, writers, actors) are instances of the Person class and have appropriate attributes.

## Example:

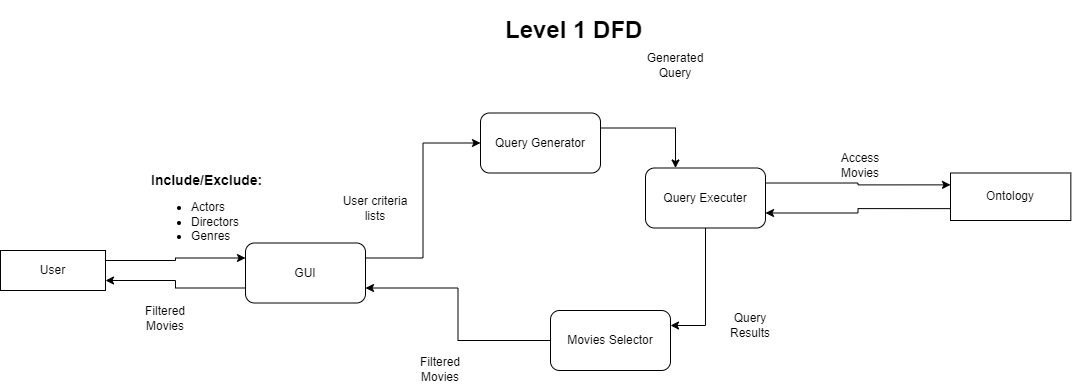
Consider the movie "The Dark Knight" released in 2008, directed by Christopher Nolan, written by Christopher Nolan and Jonathan Nolan, starring Christian Bale, Heath Ledger, and Aaron Eckhart. It falls under the genres of Action, Crime, and Drama.

# Data Flow Diagrams

## Context level



## Level 1



# Querying the ontology

**PREFIXES:**

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/anass/ontologies/movies#>

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

1. SELECT ?v WHERE{

?v rdf:type ont:actor .

}  


Figure 1 Querying the ontology 1

1. SELECT ?v WHERE{

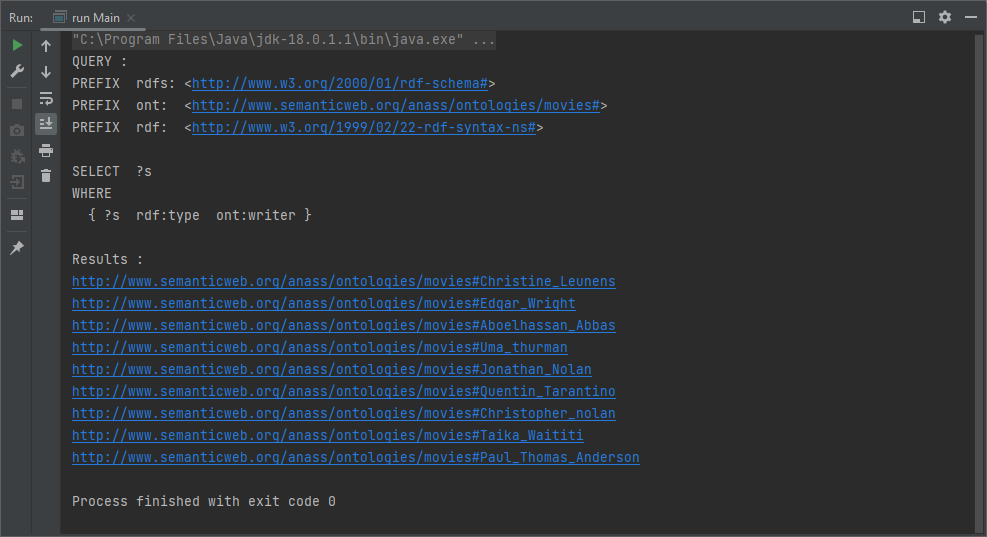
?v rdf:type ont:writer .  
}  


Figure 2 Querying the ontology 2

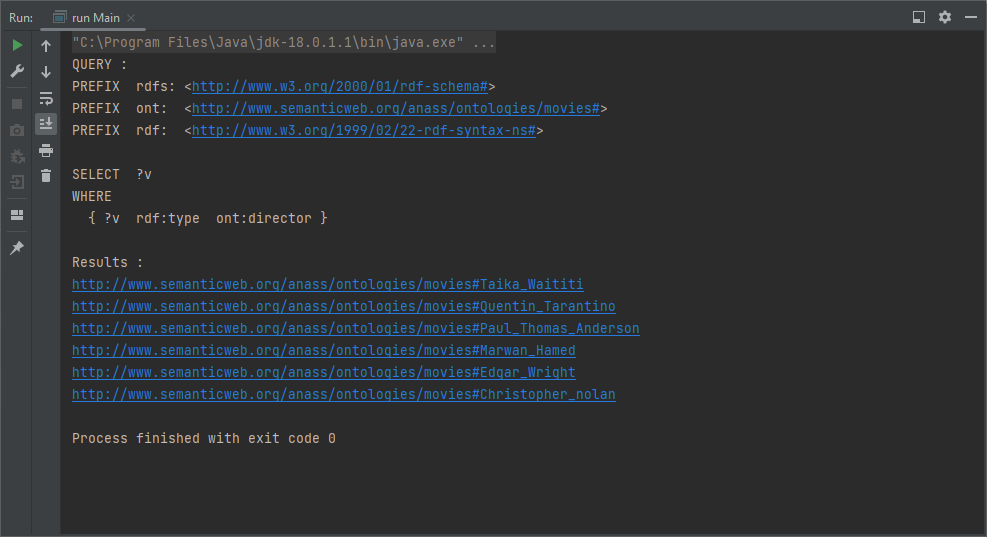
1. SELECT ?v WHERE{  
   ?v rdf:type ont:director .  
   }  
   

Figure 3 Querying the ontology 3

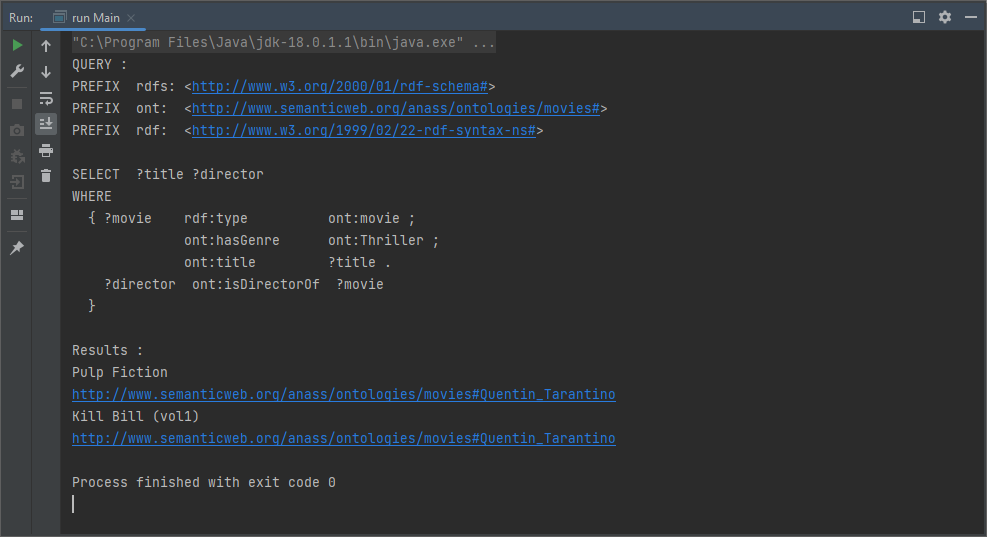
1. SELECT ?title ?director WHERE{  
   ?movie rdf:type ont:movie .  
   ?movie ont:hasGenre ont:Thriller .  
   ?movie ont:title ?title .  
   ?movie ont:hasDirector ?director .  
   }  
   

Figure 4 Querying the ontology 4

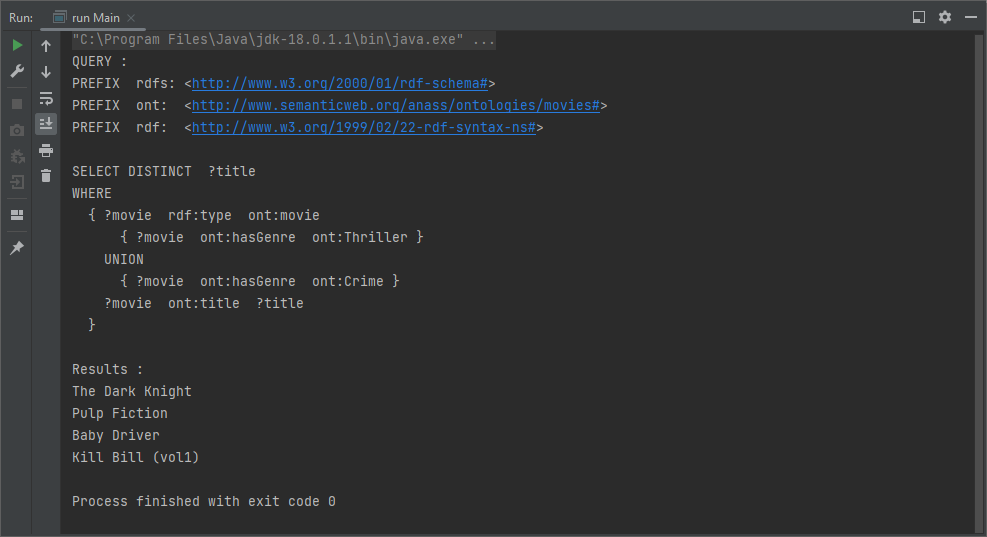
1. SELECT DISTINCT ?title WHERE{  
   ?movie rdf:type ont:movie .  
   {?movie ont:hasGenre ont:Thriller .}  
   UNION  
   {?movie ont:hasGenre ont:Crime .}  
   ?movie ont:title ?title .  
   }  
   

Figure 5 Querying the ontology 5

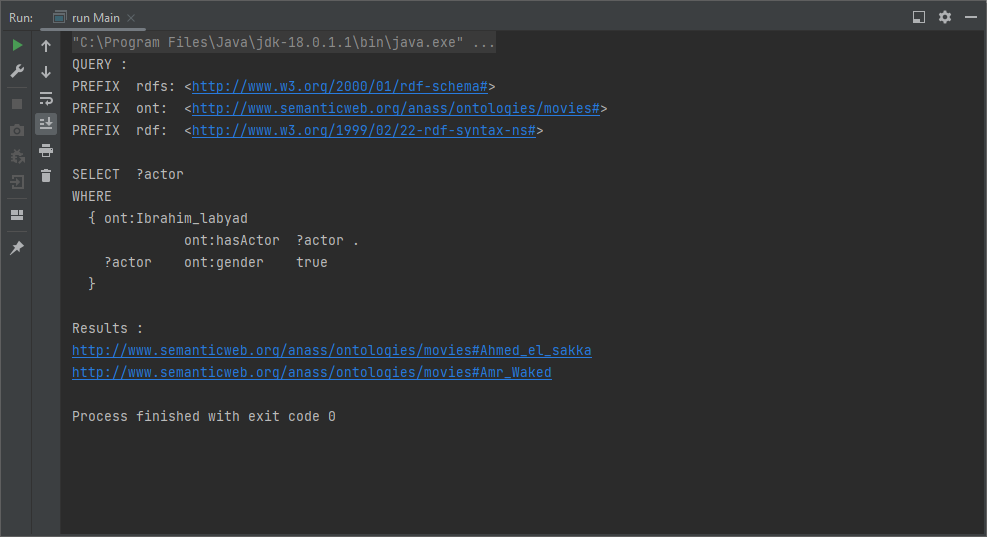
1. SELECT ?actor WHERE{  
   ont:Ibrahim\_labyad ont:hasActor ?actor .  
   ?actor ont:gender true .  
   }  
   

Figure 6 Querying the ontology 6

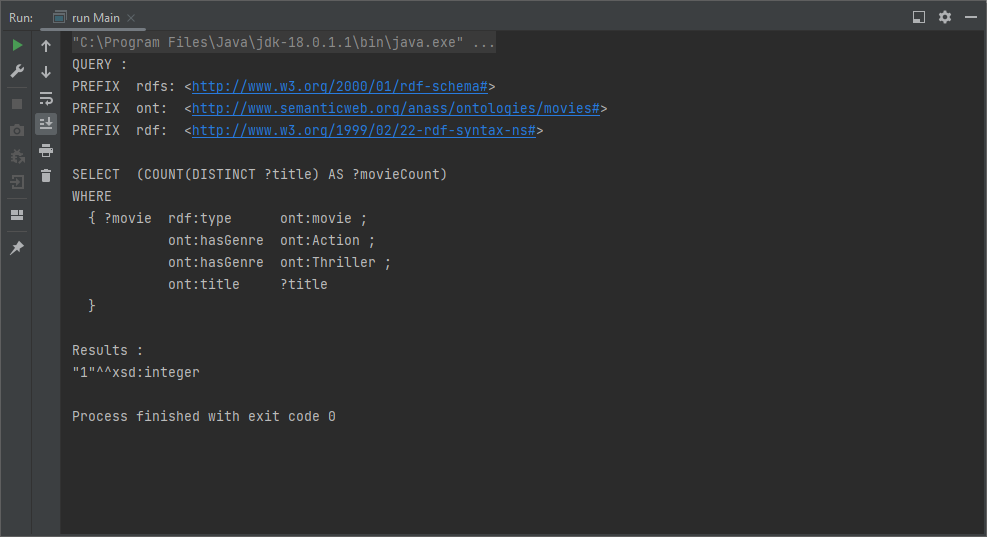
1. SELECT (COUNT(DISTINCT(?title)) AS ?movieCount) WHERE{  
   ?movie rdf:type ont:movie .  
   ?movie ont:hasGenre ont:Action .  
   ?movie ont:hasGenre ont:Thriller .  
   ?movie ont:title ?title .  
   }  
   

Figure 7 Querying the ontology 7

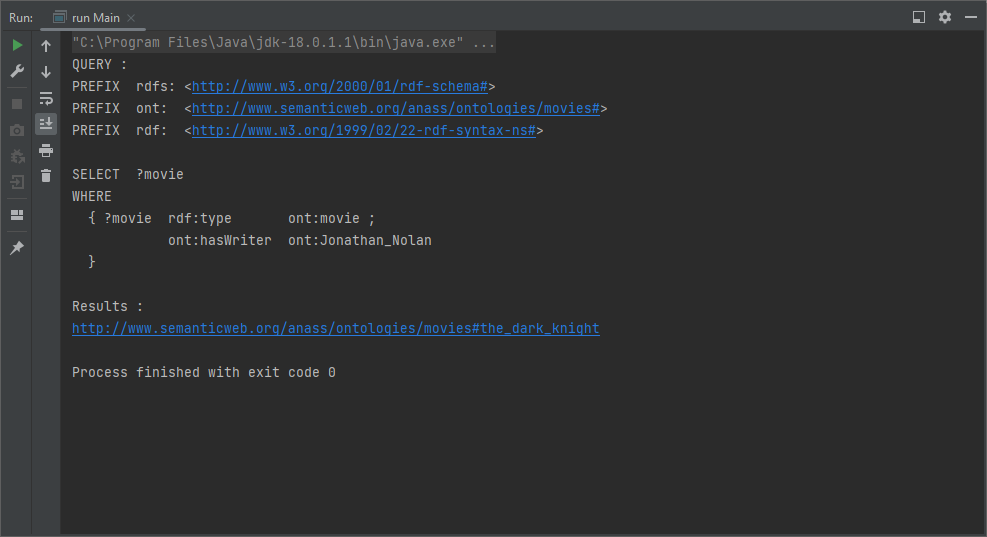
1. SELECT ?movie WHERE{  
   ?movie rdf:type ont:movie .  
   ?movie ont:hasWriter ont:Jonathan\_Nolan .  
   }  
   

Figure 8 Querying the ontology 8

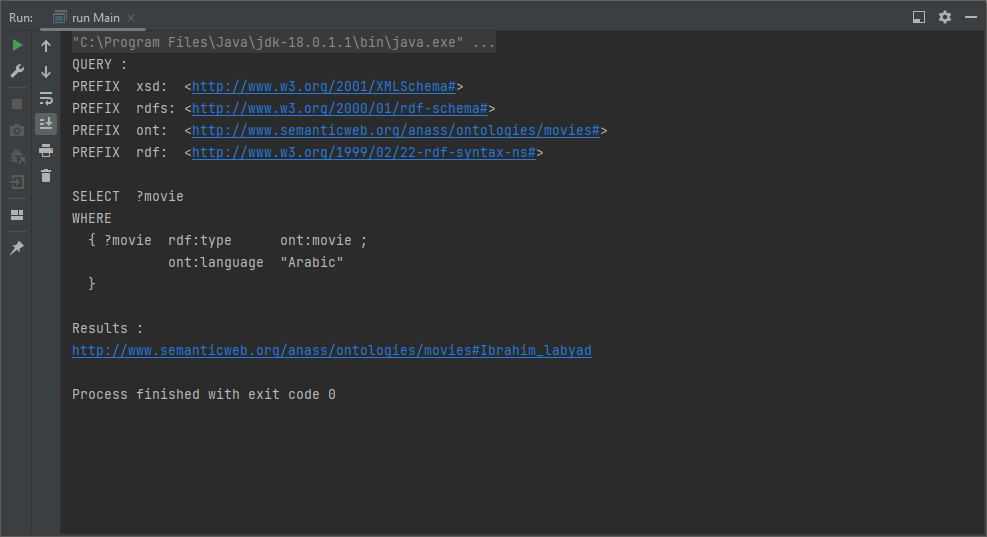
1. SELECT ?movie WHERE{  
   ?movie rdf:type ont:movie .  
   ?movie ont:language "English" .  
   }  
     
   

Figure 9 Querying the ontology 9

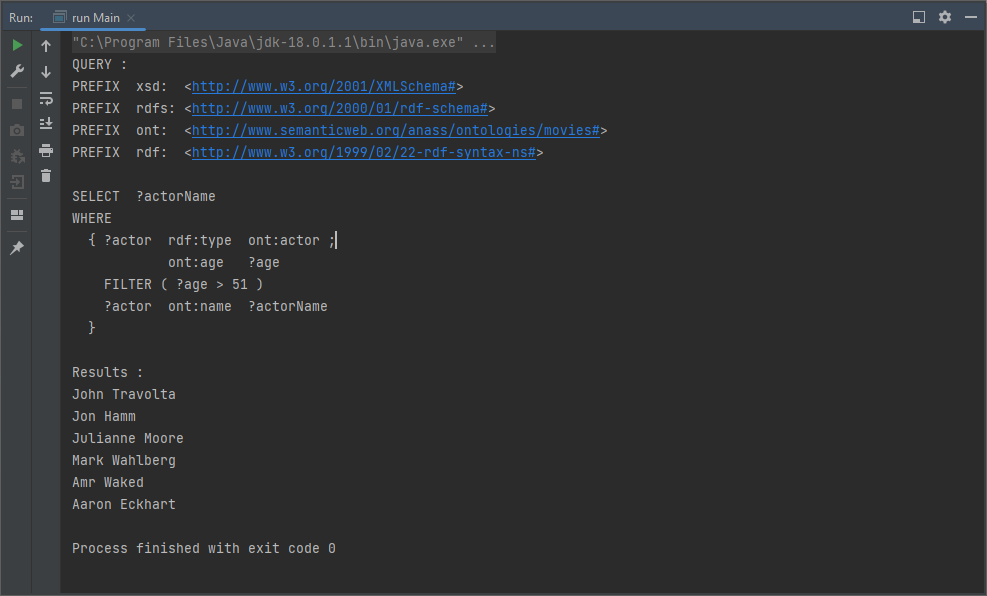
1. SELECT ?actorName WHERE{  
   ?actor rdf:type ont:actor .  
   ?actor ont:age ?age .  
   FILTER(?age>51)  
   ?actor ont:name ?actorName .  
   }  
     
   

Figure 10 Querying the ontology 10

**Propose 10 SPARQL:**

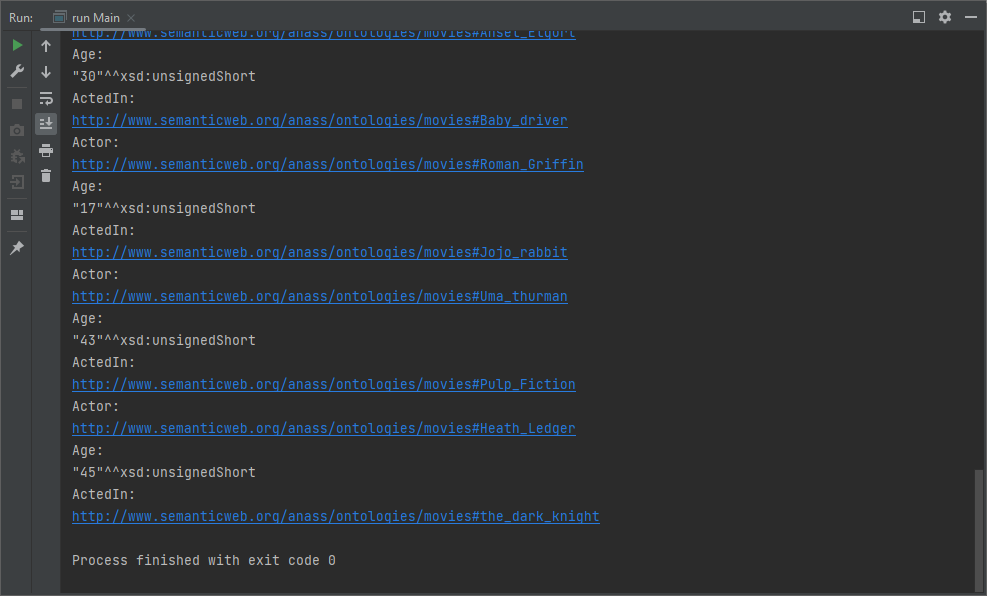
1. SELECT ?actor ?actorAge ?actedIn WHERE{  
   ?actor rdf:type ont:actor .  
   OPTIONAL { ?actor ont:isActorOf ?actedIn . }  
   OPTIONAL { ?actor ont:age ?actorAge . }  
   }  
   

Figure 11 SPARQL query 1

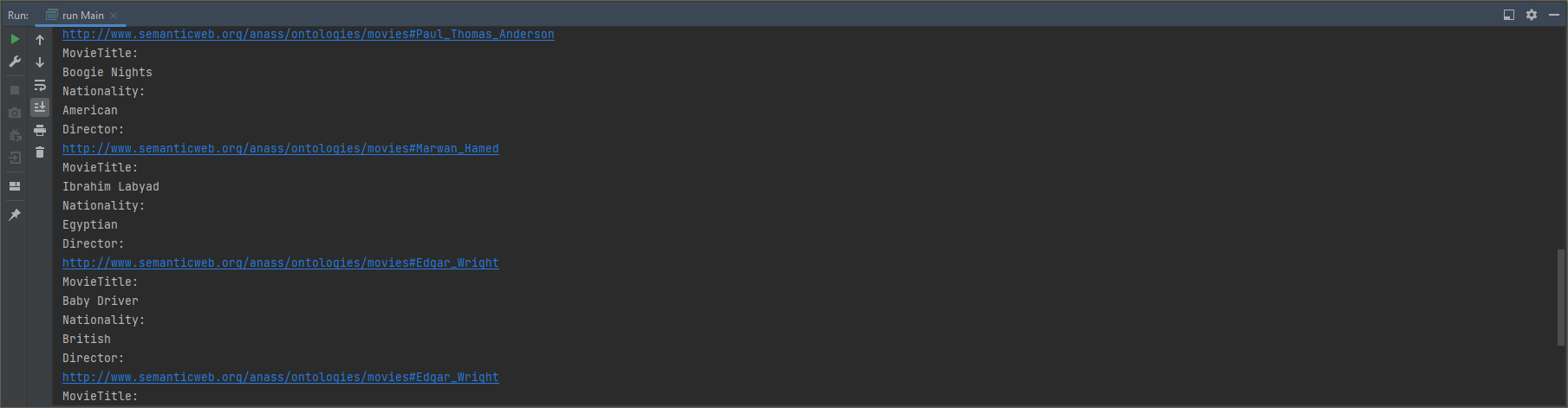
1. SELECT ?director ?movieTitle ?nationality WHERE{  
   ?director rdf:type ont:director .  
   ?director ont:isDirectorOf ?movie .  
   ?movie ont:title ?movieTitle .  
   OPTIONAL { ?director ont:isDirectorOf ont:Ibrahim\_labyad . }  
   ?director ont:nationality ?nationality .  
   OPTIONAL { ?director ont:nationality "Egyptian" . }  
   }  
   

Figure 12 SPARQL query 2

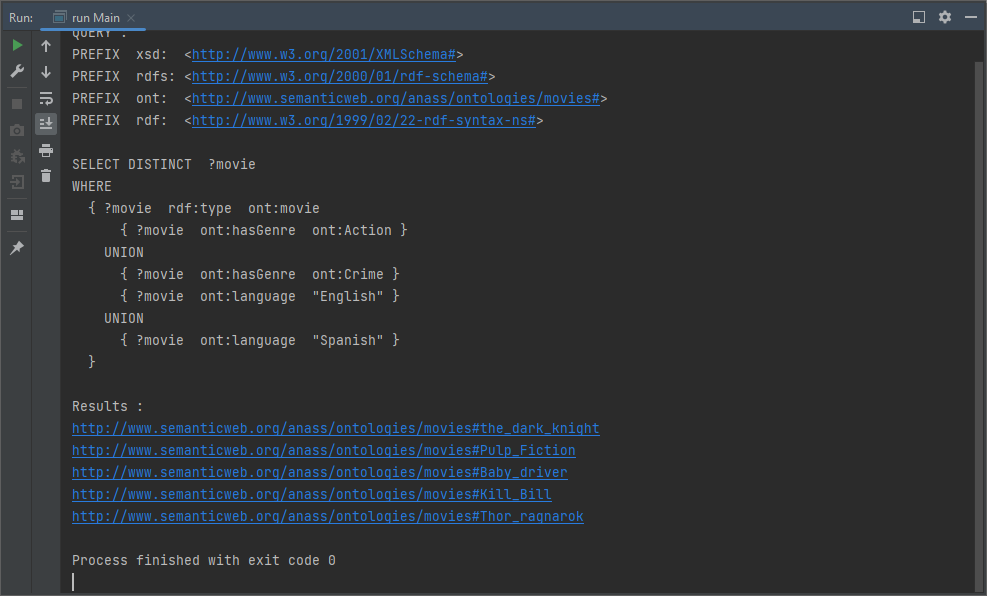
1. SELECT DISTINCT ?movie WHERE{  
   ?movie rdf:type ont:movie .  
   { ?movie ont:hasGenre ont:Action . }  
   UNION  
   { ?movie ont:hasGenre ont:Crime . }  
   { ?movie ont:language "English" . }  
   UNION  
   { ?movie ont:language "Spanish" . }  
   }  
   

Figure 13 SPARQL query 3

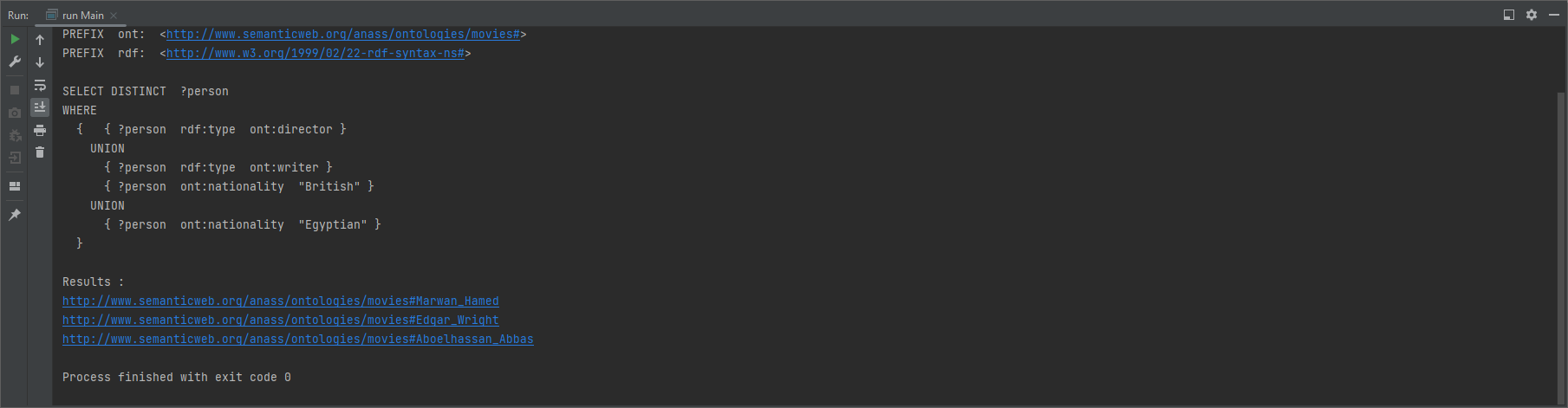
1. SELECT DISTINCT ?person WHERE{  
   { ?person rdf:type ont:director . }  
   UNION  
   { ?person rdf:type ont:writer . }  
   { ?person ont:nationality "British" . }  
   UNION  
   { ?person ont:nationality "Egyptian" . }  
   }  
   

Figure 14 SPARQL query 4

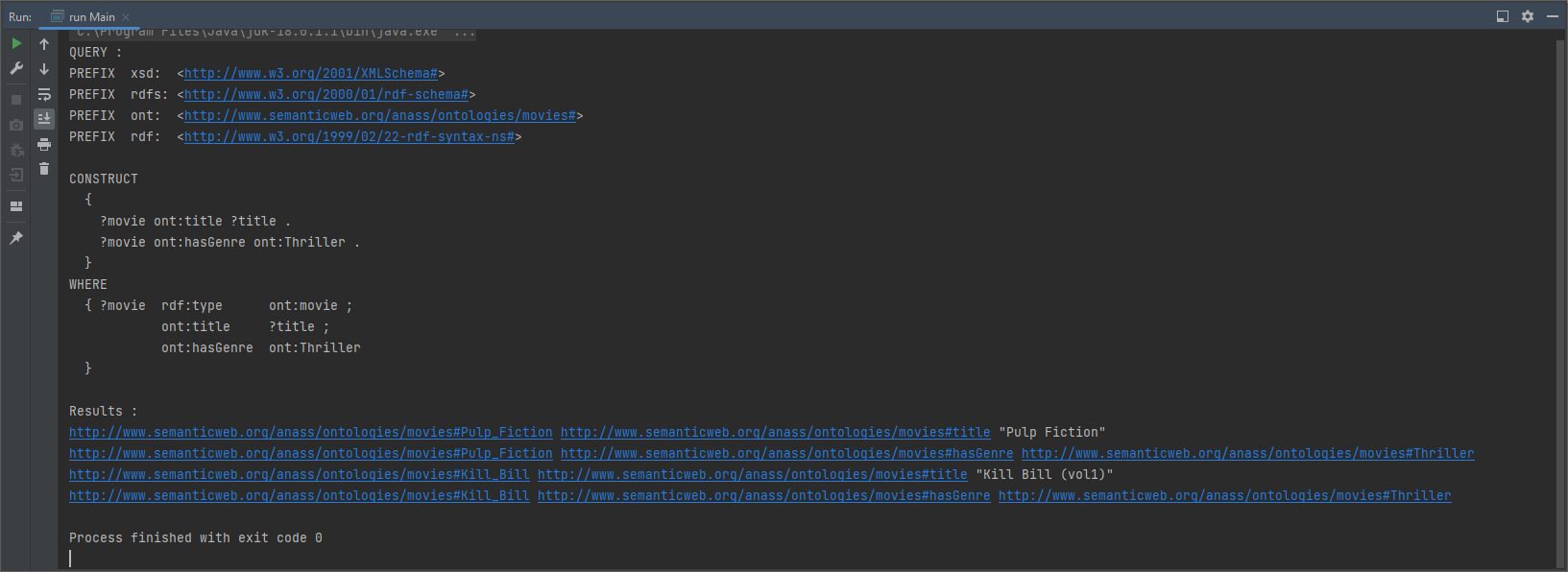
1. CONSTRUCT{  
   ?movie ont:title ?title .  
   ?movie ont:hasGenre ont:Thriller .  
   }  
   WHERE{  
   ?movie rdf:type ont:movie .  
   ?movie ont:title ?title .  
   ?movie ont:hasGenre ont:Thriller .  
   }  
   

Figure 15 SPARQL query 5

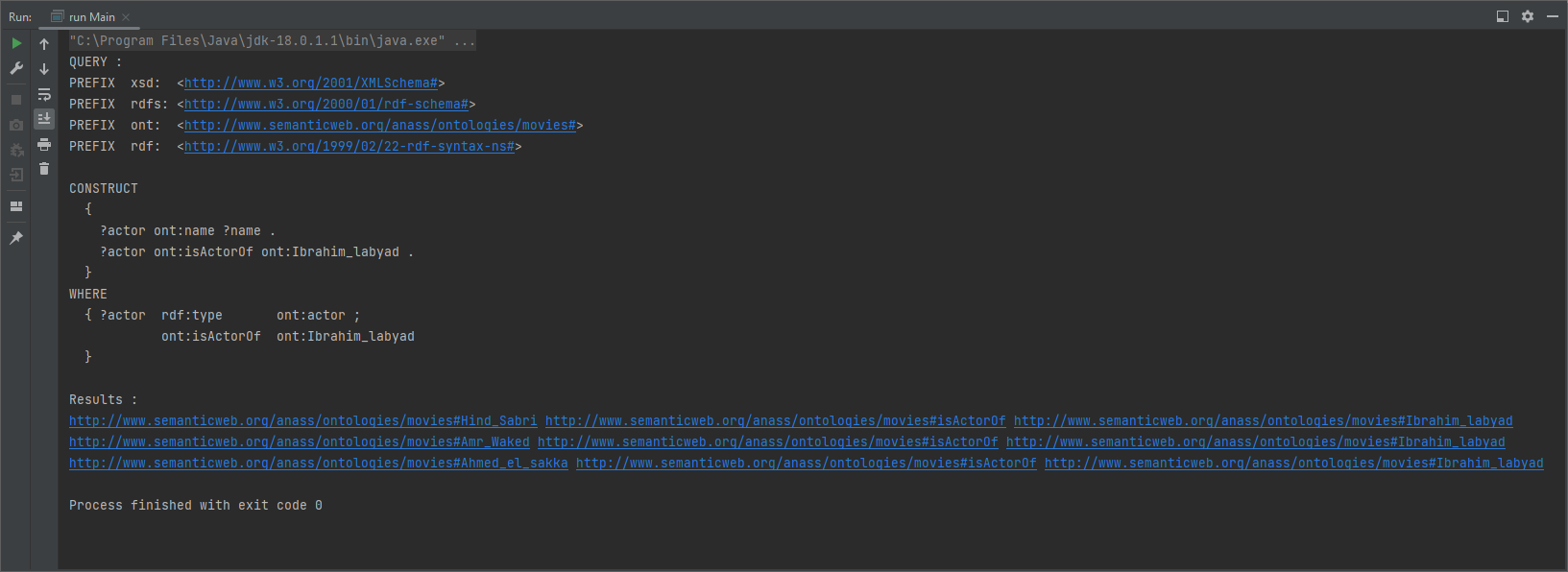
1. CONSTRUCT{  
   ?actor ont:name ?name .  
   ?actor ont:isActorOf ont:Ibrahim\_labyad .  
   }  
   WHERE{  
   ?actor rdf:type ont:actor .  
   ?actor ont:isActorOf ont:Ibrahim\_labyad .  
   }  
   

Figure 16 SPARQL query 6

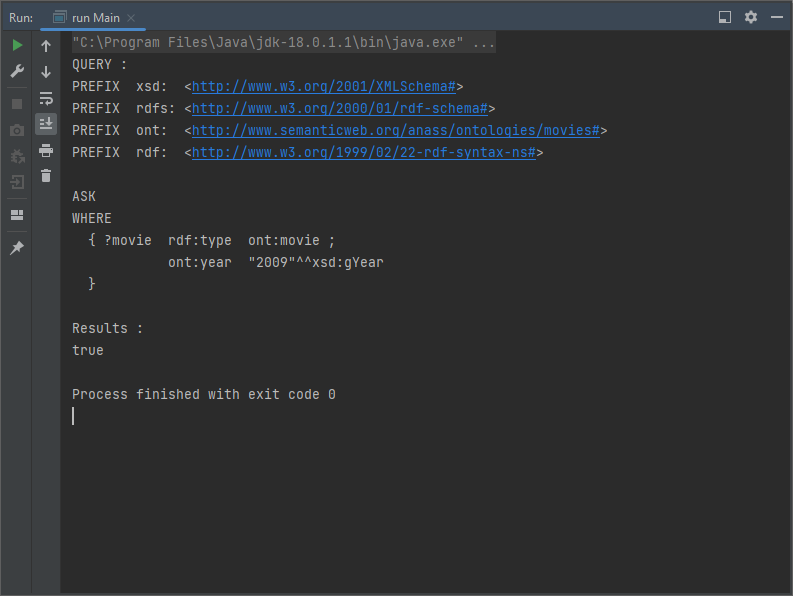
1. ASK  
   WHERE{  
   ?movie rdf:type ont:movie .  
   ?movie ont:year "2009"^^xsd:gYear .  
   }  
     
   

Figure 17 SPARQL query 7

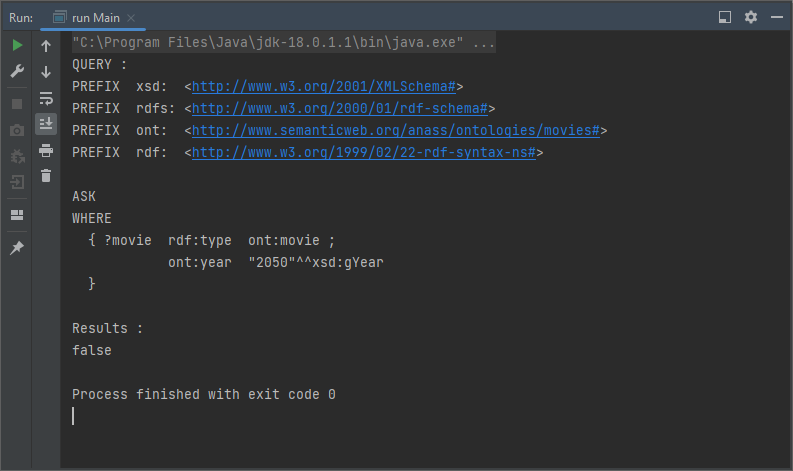
1. ASK  
   WHERE{  
   ?movie rdf:type ont:movie .  
   ?movie ont:year "2050"^^xsd:gYear .  
   }  
   

Figure 18 SPARQL query 8

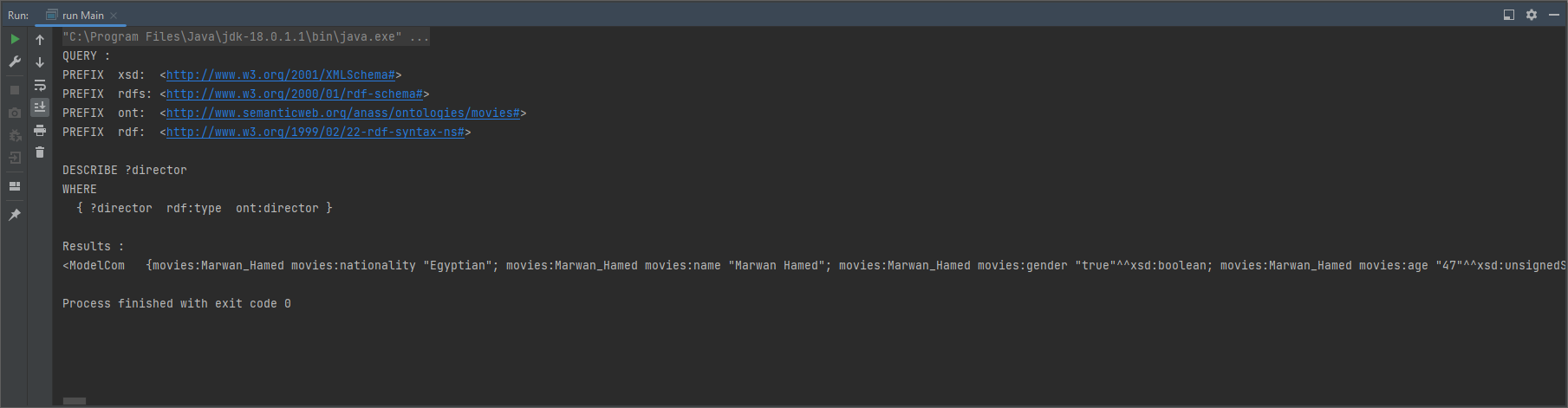
1. DESCRIBE ?director WHERE{  
   ?director rdf:type ont:director .  
   }  
   

Figure 19 SPARQL query 9

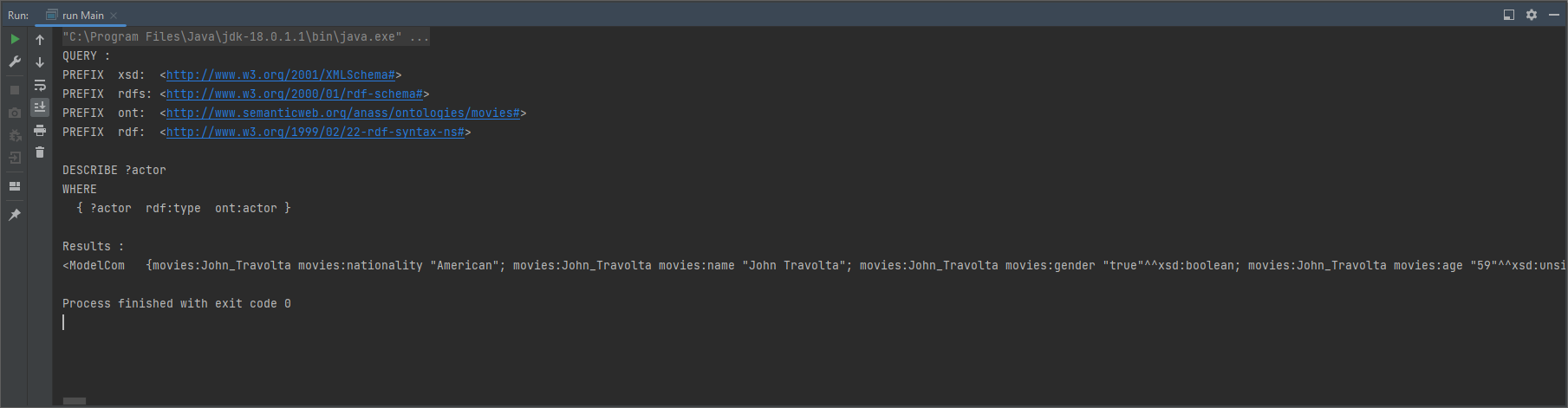
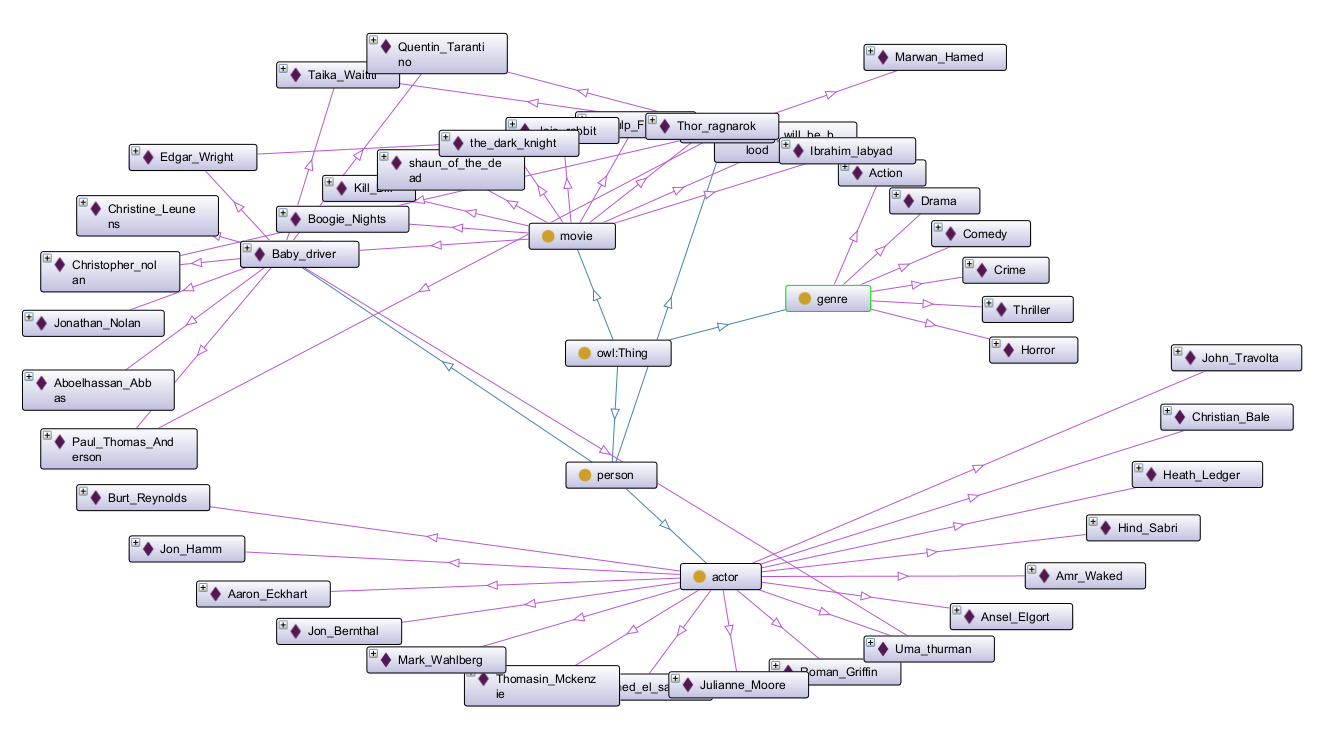
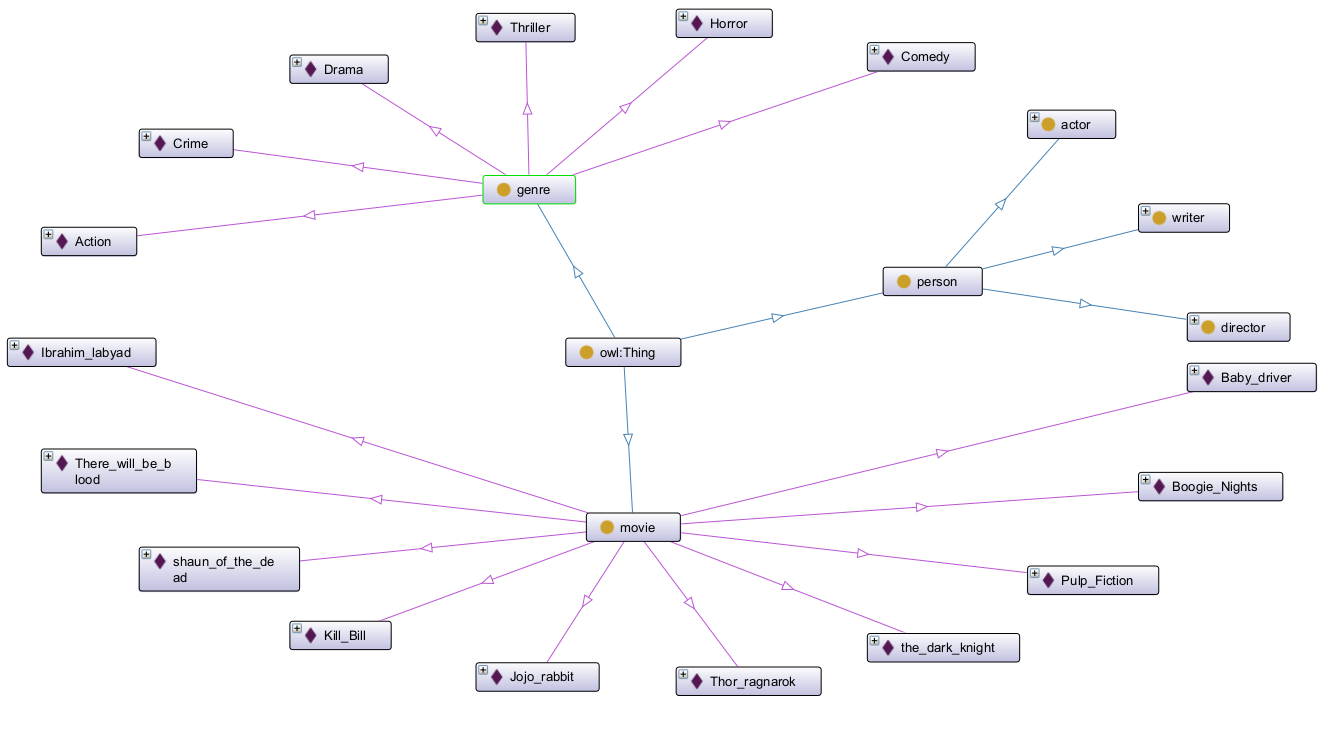
1. DESCRIBE ?actor WHERE{  
   ?actor rdf:type ont:actor .  
   }  
     
   

Figure 20 SPARQL query 10

# Ontology Visualization





# GitHub link

<https://github.com/anass-zikry/Movies-Ontology>

# Part 5 application

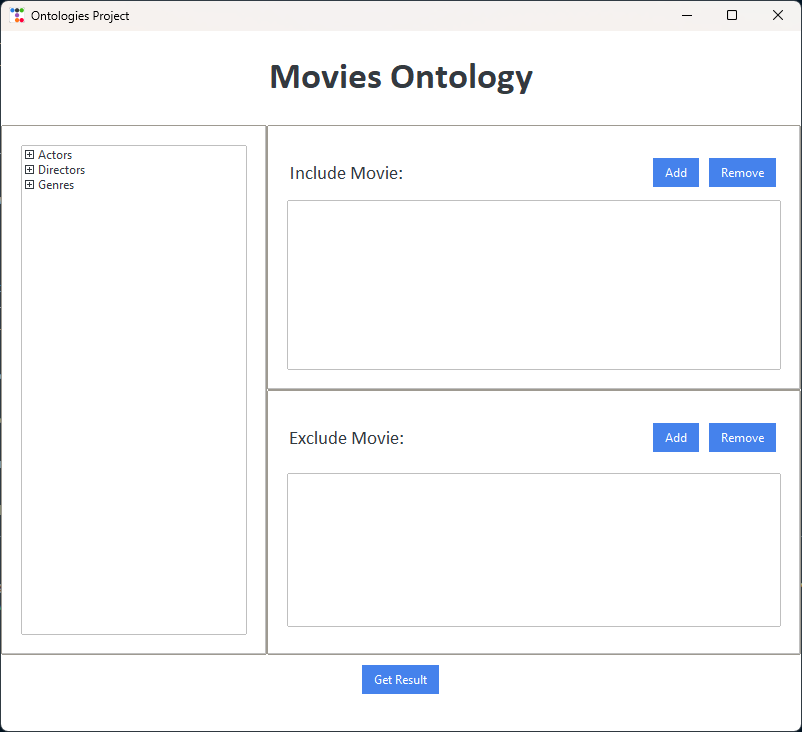
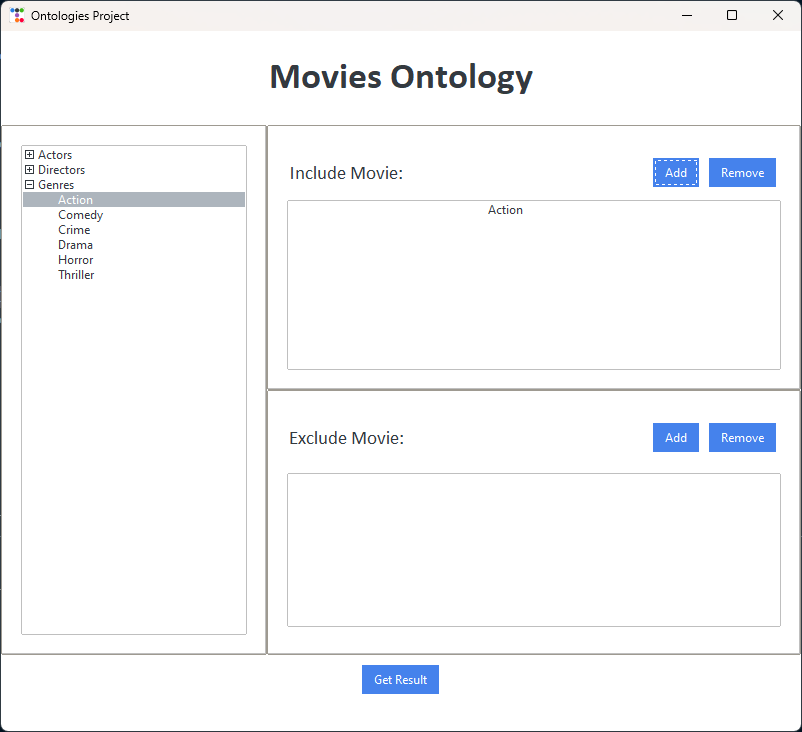
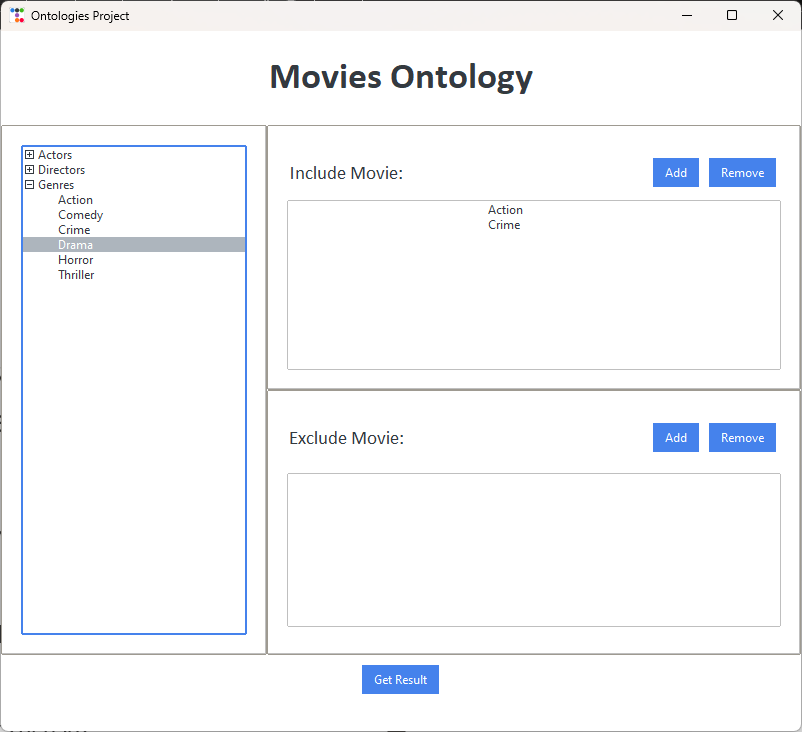
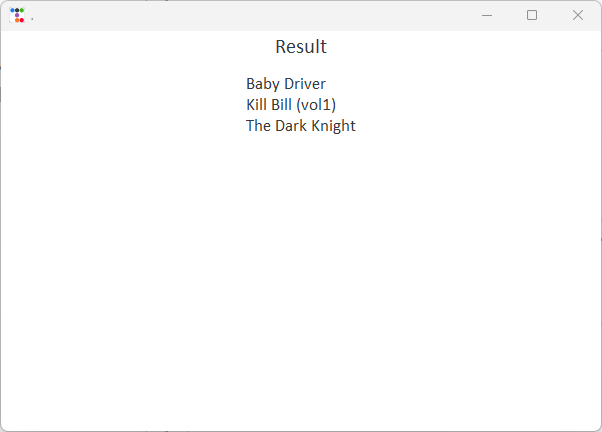
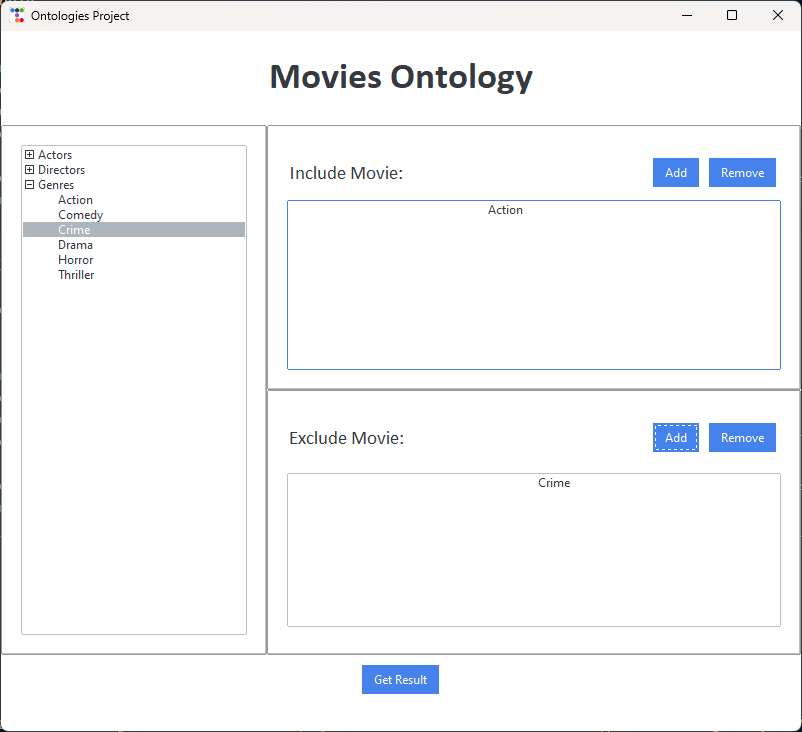


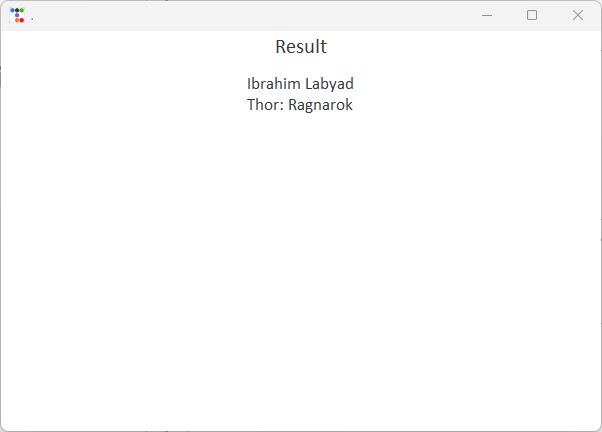
Figure 21 app main screen

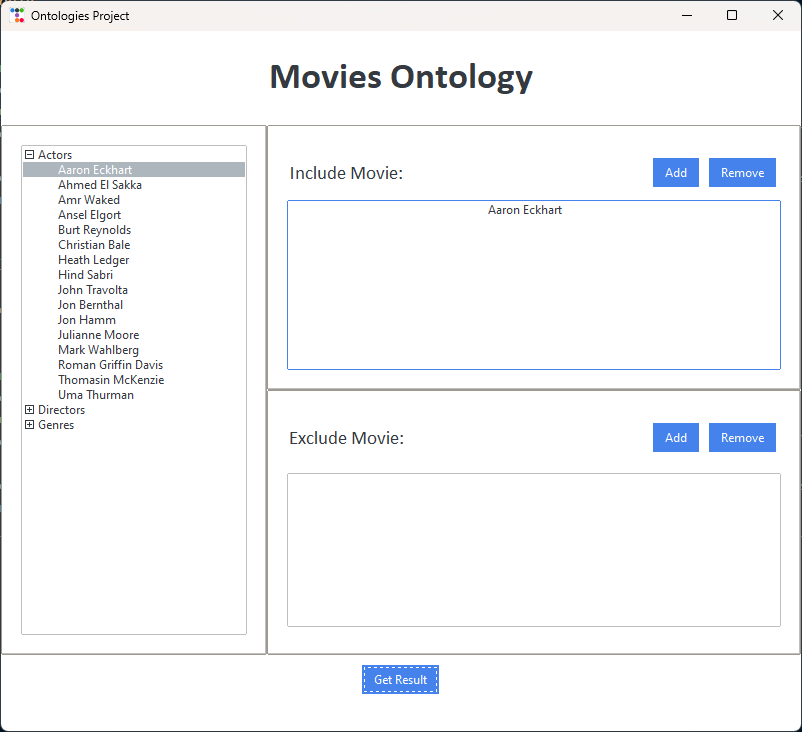


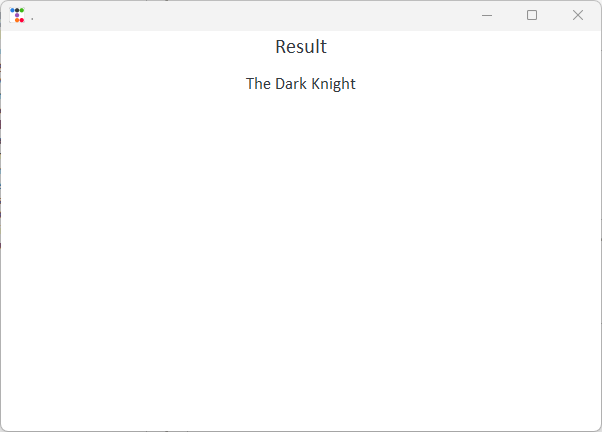












# Table of figures

[Figure 1 Querying the ontology 1 5](#_Toc166537685)

[Figure 2 Querying the ontology 2 6](#_Toc166537686)

[Figure 3 Querying the ontology 3 7](#_Toc166537687)

[Figure 4 Querying the ontology 4 8](#_Toc166537688)

[Figure 5 Querying the ontology 5 9](#_Toc166537689)

[Figure 6 Querying the ontology 6 9](#_Toc166537690)

[Figure 7 Querying the ontology 7 10](#_Toc166537691)

[Figure 8 Querying the ontology 8 11](#_Toc166537692)

[Figure 9 Querying the ontology 9 12](#_Toc166537693)

[Figure 10 Querying the ontology 10 13](#_Toc166537694)

[Figure 11 SPARQL query 1 14](#_Toc166537695)

[Figure 12 SPARQL query 2 15](#_Toc166537696)

[Figure 13 SPARQL query 3 16](#_Toc166537697)

[Figure 14 SPARQL query 4 17](#_Toc166537698)

[Figure 15 SPARQL query 5 17](#_Toc166537699)

[Figure 16 SPARQL query 6 18](#_Toc166537700)

[Figure 17 SPARQL query 7 19](#_Toc166537701)

[Figure 18 SPARQL query 8 20](#_Toc166537702)

[Figure 19 SPARQL query 9 20](#_Toc166537703)

[Figure 20 SPARQL query 10 21](#_Toc166537704)

[Figure 21 app main screen 23](#_Toc166537705)