Lecture 7

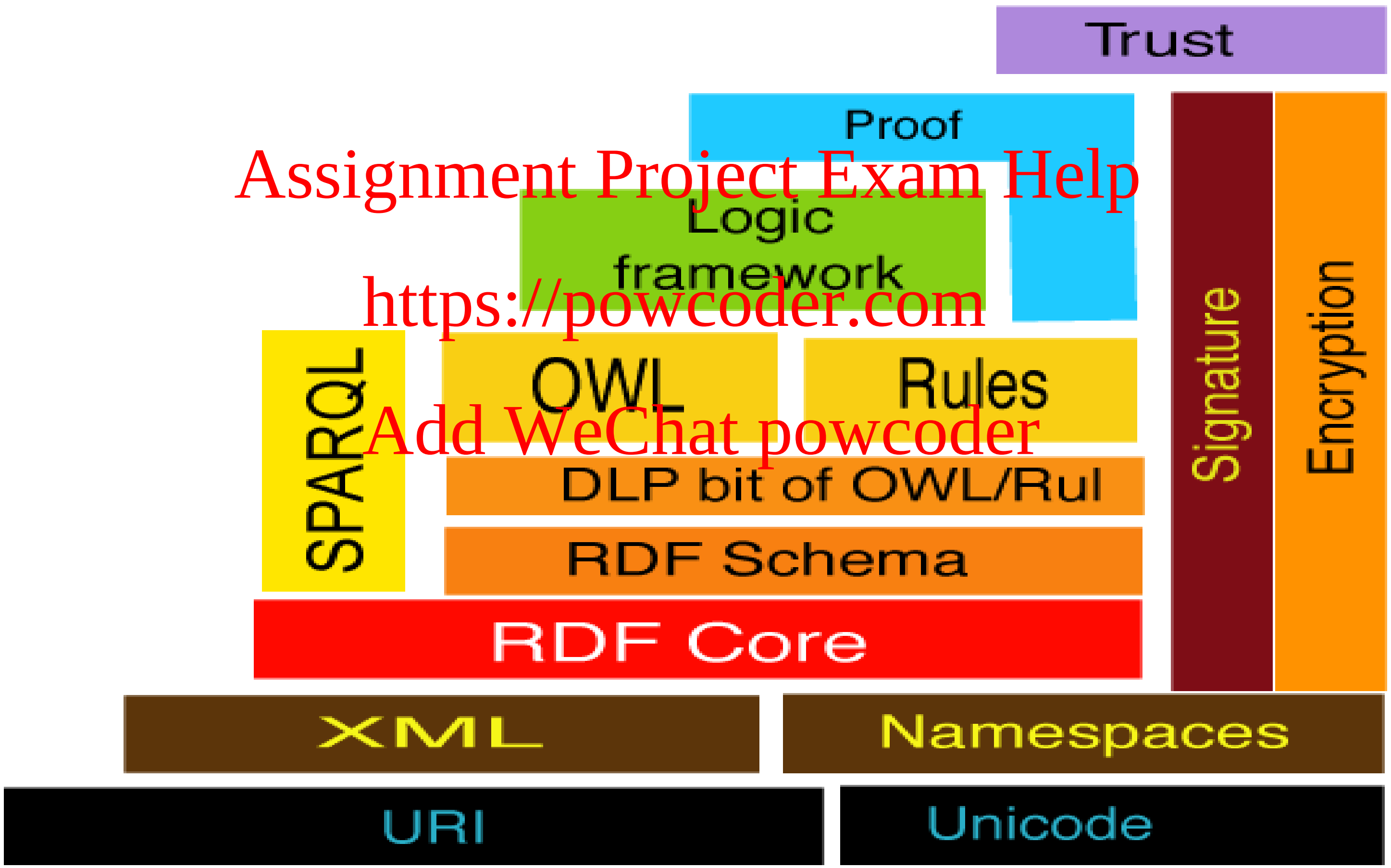
Chapter 4

Web Ontology Language: OWL

Grigoris Antoniou

Frank van Harmelen

A Semantic Web Layer Stack



- A reasoner expands the number of triples based on relations like rdfs:subClassOf, rdfs:range, rdfs:domain, etc.

In this lecture you will learn about OWL:

- OWL adds semantics to the schema.

- With OWL you can add more about classes and properties.

- OWL has the ability to say two entities are the same, useful when merging datasets.

Lecture Outline

1. **Basic Ideas of OWL**
2. The OWL Language
3. Examples
4. The OWL Namespace
5. Future Extensions

Requirements for Ontology Languages

- Ontology languages allow users to write explicit, formal conceptualizations of domain models
- There are five requirements with any Ontology language:
  I. a well-defined syntax
  II. efficient reasoning support
  III. a formal semantics
  IV. sufficient expressive power
  V. convenience of expression

Tradeoff between Expressive Power and Efficient Reasoning Support

- The richer the language is, the more inefficient the reasoning support.

Therefore, we need a compromise:

- On the one hand, a language supported by reasonably efficient reasoners.
- On the other hand, a language that can express large classes of ontologies and knowledge.

Why does efficient reasoning conflict with expressiveness?

Reasoning About Knowledge in Ontology Languages

- Class membership
  - If x is an instance of a class C, and C is a subclass of D, then we can infer that x is an instance of D
- Equivalence of classes
  - If class A is equivalent to class B, and class B is equivalent to class C, then A is equivalent to C, too

Reasoning About Knowledge in Ontology Languages (2)

- Consistency
  - X instance of classes A and B, but A and B are disjoint
  - This is an indication of an error in the ontology
- Classification
  - Certain property-value pairs are a sufficient condition for membership in a class A; if an individual x satisfies such conditions, we can conclude that x must be an instance of A

If it **looks like a duck**, swims **like a duck**, and quacks **like a duck**, then it probably is a **duck**.

Uses for Reasoning

- Reasoning support is important for
  - checking the consistency of the ontology and the knowledge
  - checking for unintended relationships between classes
  - automatically classifying instances in classes
- Checks like the preceding ones are valuable for
  - designing large ontologies, where multiple authors are involved
  - integrating and sharing ontologies from various sources

Reasoning Support for OWL

- Semantics is a prerequisite for reasoning support
- Formal semantics and reasoning support are usually provided by
  - mapping an ontology language to a known logical formalism
  - using automated reasoners that already exist for those formalisms
- OWL is (partially) mapped on a description logic, and makes use of reasoners such as Pellet, FaCT and RACER
- Description logics are a subset of predicate logic for which efficient reasoning support is possible

Three Species of OWL

- W3C'sWeb Ontology Working Group defined OWL as three different sublanguages:
  - OWL Full
  - OWL DL
  - OWL Lite
- Each sublanguage geared toward fulfilling different aspects of requirements

OWL Full

- uses all the OWL languages primitives
- allows the combination of these primitives in arbitrary ways with RDF and RDF Schema
- OWL Full is fully upward-compatible with RDF, both syntactically and semantically
- OWL Full is is undecidable
  – No complete (or efficient) reasoning support

OWL DL

- OWL DL (Description Logic) is a sublanguage of OWL Full
- OWL DL permits efficient reasoning support
- **But** we lose full compatibility with RDF:
  - Not every RDF document is a legal OWL DL document.
  - Every legal OWL DL document is a legal RDF document.

OWL Lite

- Easier to
  - grasp, for users
  - implement, for tool builders
- Restricted expressivity

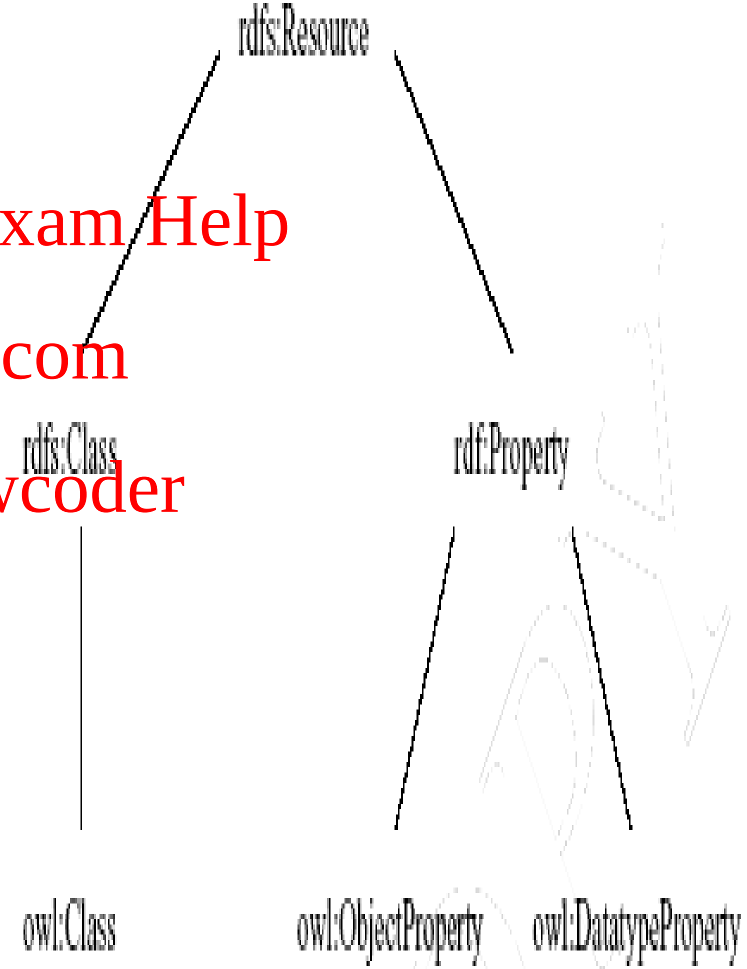
Upward Compatibility between OWL Species

- Every legal OWL Lite ontology is a legal OWL DL ontology

- Every legal OWL DL ontology is a legal OWL Full ontology

A Figure showing OWL Compatibility with RDF Schema

- All varieties of OWL use RDF for their syntax
- Instances are declared as in RDF, using RDF descriptions



Lecture Outline

1. Basic Ideas of OWL
2. **The OWL Language**
3. Examples
4. The OWL Namespace
5. Future Extensions

OWL Syntactic Varieties

- OWL builds on RDF and uses RDF's XML-based syntax
- Other syntactic forms for OWL have also been defined:
  - An alternative, more readable XML-based syntax
  - An abstract syntax, that is much more compact and readable than the XML languages
  - A graphic syntax based on the conventions of UML

OWL XML/RDF Syntax: Header

```
<rdf:RDF
    xmlns:owl ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-
    syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-
    schema#"
    xmlns:xsd ="http://www.w3.org/2001/
    XLMSchema#">
```

owl:Ontology

```
<owl:Ontology rdf:about="">
    <rdfs:comment>An example OWL ontology </rdfs:comment>
    <owl:priorVersion
            rdf:resource="http://www.mydomain.org/uni-ns-old"/>
    <owl:imports rdf:resource="http://www.mydomain.org/persons"/>
    <rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>
```


- **owl:imports** is **a transitive property**
- **about=""** **means the xml:base (current document), show in the next slide**

```
<rdf:RDF
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xml:base="http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl">
```


Source:https://protegewiki.stanford.edu/wiki/How_Owl_Imports_Work

<http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl> **owl:imports** **<http://protege.stanford.edu/plugins/owl/protege>.**

This statement tells tools such as Protege or reasoners that the pizza ontology should import an ontology whose name is **http://protege.stanford.edu/plugins/owl/protege.**

Source:https://protegewiki.stanford.edu/wiki/How_Owl_Imports_Work

Classes

- Classes are defined using **owl:Class**
  - owl:Class is a subclass of rdfs:Class

- Disjointness is defined using **owl:disjointWith**


```
<owl:Class rdf:about="#associateProfessor">
    <owl:disjointWith rdf:resource="#professor"/>
    <owl:disjointWith
    rdf:resource="#assistantProfessor"/>
</owl:Class>
```

Classes (2)

- **owl:equivalentClass** defines equivalence of classes

```
<owl:Class rdf:ID="faculty">
  <owl:equivalentClass rdf:resource=
  "#academicStaffMember"/>
</owl:Class>
```


- **owl:Thing** is the most general class, which contains everything
- **owl:Nothing** is the empty class

Properties

- **In OWL there are two kinds of properties**
  - **Object properties, which relate objects to other objects**
    - **E.g. is-TaughtBy, supervises**
  - **Data type properties, which relate objects to datatype values**
    - **E.g. phone, title, age, etc.**

Datatype Properties

- OWL makes use of XML Schema data types, using the layered architecture of the Semantic Web


```
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:range rdf:resource=
  "http://www.w3.org/2001/XLMSchema
        #nonNegativeInteger"/>
</owl:DatatypeProperty>
```

Object Properties

- User-defined data types


```
<owl:ObjectProperty rdf:ID="isTaughtBy">
    <owl:domain rdf:resource="#course"/>
    <owl:range rdf:resource=
    "#academicStaffMember"/>
    <rdfs:subPropertyOf rdf:resource="#involves"/>
</owl:ObjectProperty>
```

Inverse Properties

```
<owl:ObjectProperty rdf:ID="teaches">
    <rdfs:range rdf:resource="#course"/>
    <rdfs:domain rdf:resource="#academicStaffMember"/>
    <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```

Equivalent Properties

owl:equivalentProperty

```
<owl:ObjectProperty rdf:ID="lecturesIn">
<owl:equivalentProperty rdf:resource="#teaches"/>
</owl:ObjectProperty>
```

Property Restrictions

- In OWL we can declare that the class C satisfies certain conditions
  - All instances of C satisfy the conditions
- This is equivalent to saying that C is subclass of a class C', where C' collects all objects that satisfy the conditions
  - C' can remain anonymous

Property Restrictions (2)

- A (restriction) class is achieved through an **owl:Restriction** element
- This element contains **an owl:onProperty** element and **one or more restriction declarations**
- One type defines **cardinality restrictions** (at least one, at most 3,…)

Property Restrictions (3)

- The other type defines restrictions on the kinds of values the property may take
  - **owl:allValuesFrom** **specifies universal quantification**

    ∀ (the universal quantifier symbol)
  - **owl:hasValue** must have at least one value X
  - **owl:someValuesFrom** specifies existential quantification

    ∃ (read: “there **exists**”).

owl:allValuesFrom

Write the equivalence in English

```
<owl:Class rdf:about="#firstYearCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
          <owl:onProperty rdf:resource="#isTaughtBy"/>
          <owl:allValuesFrom rdf:resource="#Professor"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

ANSWER

Only professors teach first-year subjects

owl:hasValue

Write the equivalence in English

```
<owl:Class rdf:about="#mathCourse">
        <rdfs:subClassOf>
        <owl:Restriction>
        <owl:onProperty rdf:resource= "#isTaughtBy"/>
            <owl:hasValue rdf:resource= "#949352"/>
        </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

Which one is correct?

1) All of the math courses must have a value #949352 for their isTaugBy property.

2) One of the math courses is taught by **#949352**

3) All math courses must be taught by #949352 AND maybe others.

Cardinality Restrictions

- We can specify minimum and maximum number using **owl:minCardinality** and **owl:maxCardinality**
- It is possible to specify a precise number by using the same minimum and maximum number
- For convenience, OWL offers also **owl:cardinality**

Cardinality Restrictions (2)

```
<owl:Class rdf:about="#course">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isTaughtBy"/>
            <owl:minCardinality rdf:datatype=
                "&xsd;nonNegativeInteger">
            1
            </owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Special Properties

- **owl:TransitiveProperty** (transitive property)
  - E.g. “has better grade than”, “is ancestor of”
- **owl:SymmetricProperty** (symmetry)
  - E.g. “has same grade as”, “is sibling of”
- **owl:FunctionalProperty** defines a property that has at most one value for each object
  - E.g. “age”, “height”, “directSupervisor”
- **owl:InverseFunctionalProperty** defines a property for which two different objects cannot have the same value

Assignment Project Exam Help

Write three examples for each of transitive and symmetric properties .

https://powcoder.com

Add WeChat powcoder

ANSWER

SYMETRIC:

1]"X isMarriedTo Y" means "Y isMarriedTo X".

2]"*Z equals D*"   means "*D equals Z*"

3]" A co-worker-of B" means " B co-worker-of A"

TRANSITIVE:

1]if A implies B and B implies C, then A implies C

2]if "*Z equals D*" and "*D equals E*" means

"*Z equalls E*"

3]if "*X sub-set-of Y*" and "*Y sub-set-of Z*" means

"*X sub-set Z*"

Special Properties (2)

```
<owl:ObjectProperty rdf:ID="hasSameGradeAs">
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
        <rdf:type
    rdf:resource="&owl;SymmetricProperty"/>
        <rdfs:domain rdf:resource="#student"/>
        <rdfs:range rdf:resource="#student"/>
</owl:ObjectProperty>
```

More on OWL

https://www.w3.org/TR/2004/REC-owl-guide-20040210/

- The *complementOf* construct selects all individuals from the domain of discourse that do not belong to a certain class. Usually this refers to a very large set of individuals:

[Source : http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)


```
<owl:Class rdf:ID="ConsumableThing" />
<owl:Class rdf:ID="NonConsumableThing">
    <owl:complementOf rdf:resource="#ConsumableThing" />
</owl:Class>
```

Boolean Combinations

- We can combine classes using Boolean operations (**union**, **intersection**, **complement**)


```
<owl:Class rdf:about="#course">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:complementOf rdf:resource=
    "#staffMember"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Boolean Combinations (2)

```
<owl:Class rdf:ID="peopleAtUni">
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#staffMember"/>
        <owl:Class rdf:about="#student"/>
    </owl:unionOf>
</owl:Class>
```


- The new class is not a subclass of the union, but rather equal to the union

Boolean Combinations (3)

```
<owl:Class rdf:ID="facultyInSCIT">
    <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#faculty"/>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#belongsTo"/>
                <owl:hasValue rdf:resource= "#SCIT"/>
            </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

A Question:

Assuming #staffMember, #faculty, and #techSupportStaff” have all been defined as cllasses, by the use of nesting Boolean operators, complete the following code for defining adminStaff. Please indent statements for the sake of clarity.

<owl:Class rdf:ID="adminStaff">

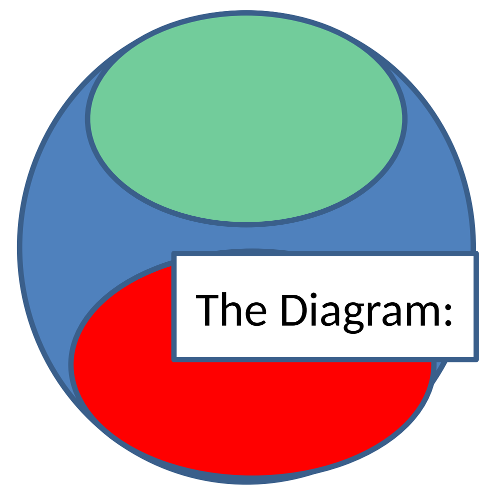
…………………………………………………

…………………………………………………

…………………………………………………

…………………………………………………

</owl:Class>



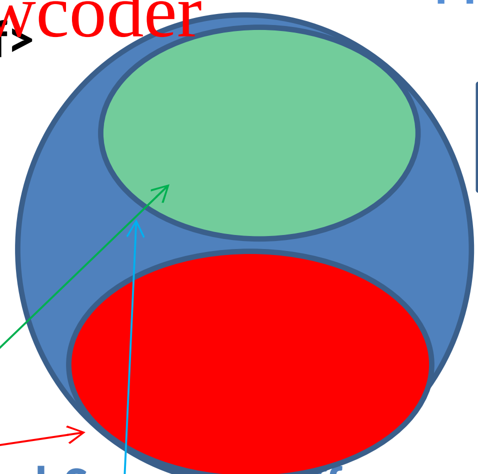
Hint:

#staffMember = #adminStaff+#faculty+#techSupportStaff

ANSWER

```
<owl:Class rdf:ID="adminStaff">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#staffMember"/>
        <owl: Class>
            <owl:complementOf>
                <owl: Class>
                    <owl:unionOf  rdf:parseType="Collection">
                        <owl:Class rdf:about="#faculty"/>
                        <owl:Class rdf:about=#techSupportStaff"/>
                    </owl:unionOf>
                </owl: Class>
            </owl:complementOf>
        </owl: Class>
    </owl:intersectionOf>
</owl:Class>
```


The Diagram:



#staffMember = #adminStaff+#faculty+#techSupportStaff

EXAMPLES:

```
<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>
```



```
<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing" />
  ...
</owl:Class>
```

EXAMPLES:

```
<Region rdf:ID="CentralCoastRegion" />
```

Note that, the same as abbreviating in RDF, the following is **identical** in meaning to the example above.


```
<owl:Thing rdf:ID="CentralCoastRegion" />
<owl:Thing rdf:about="#CentralCoastRegion">
   <rdf:type rdf:resource="#Region"/>
</owl:Thing
```

EXAMPLES:

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="course">
  <rdfs:domain rdf:resource="#Meal" />
  <rdfs:range rdf:resource="#MealCourse" />
</owl:ObjectProperty>
```

Interestingly, it is now possible to expand the definition of Wine to include the notion that a wine is made from at least one WineGrape. As with property definitions, class definitions have multiple subparts that are implicitly conjoined.

In the next slide you can see it:

[Source: https://www.w3.org/TR/owl-guide/](https://www.w3.org/TR/owl-guide/)

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
                1
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  …
</owl:Class>
```

- https://www.youtube.com/watch?v=LdsYkpFvYxU&ab_channel=MinhTr%E1%BA%A7n%C4%90%E1%BB%A9c