Lecture 4: Ontologies

TIES4520 Semantic Technologies for Developers
Autumn 2023

University of Jyväskylä

Khriyenko Oleksiy

Part 1

Ontology basics

Ontology

- A ***person's vocabulary*** is the set of words within a language that are familiar to that person. (Wikipedia)
- On the ***Semantic Web***, ***vocabularies*** define the concepts and relationships used to describe and represent an area of concern. (W3C). Vocabularies are used to:
  - *classify* the *terms* that can be used in a particular application,
  - *characterize relationships*, and
  - *define constraints* on using those terms.

Ontology

- An ***Ontology*** is an *explicit*, *formal specification of a shared conceptualization*. Ontologies are formal models that describe a certain domain and specify the definitions of terms by describing their relationships with other terms in the ontology.
- Example: medical ontology, IT ontology, music ontology, etc.
- Consists of:
  - TBox
    - Describes abstract concepts (*Class*) and their relationships (*Property*)
    - Taxonomy, classification
  - ABox
    - Describes concrete individuals (*Instance*) and their relationships to other individuals and/or abstract concepts from Tbox
- There cannot be a global ontology of everything
  - Ontologies are dynamic (they change in time)
  - Every person can have a different perspective on the domain

Instance vs. class

- **Class** (type)
  - Represents a set of things that share same properties (and/or behavior)
  - characterized via *attributes* (name-value pairs)
  - Example: Person, Fruit, Feeling…

Usually names start with a capital letter

Usually names start with a small letter

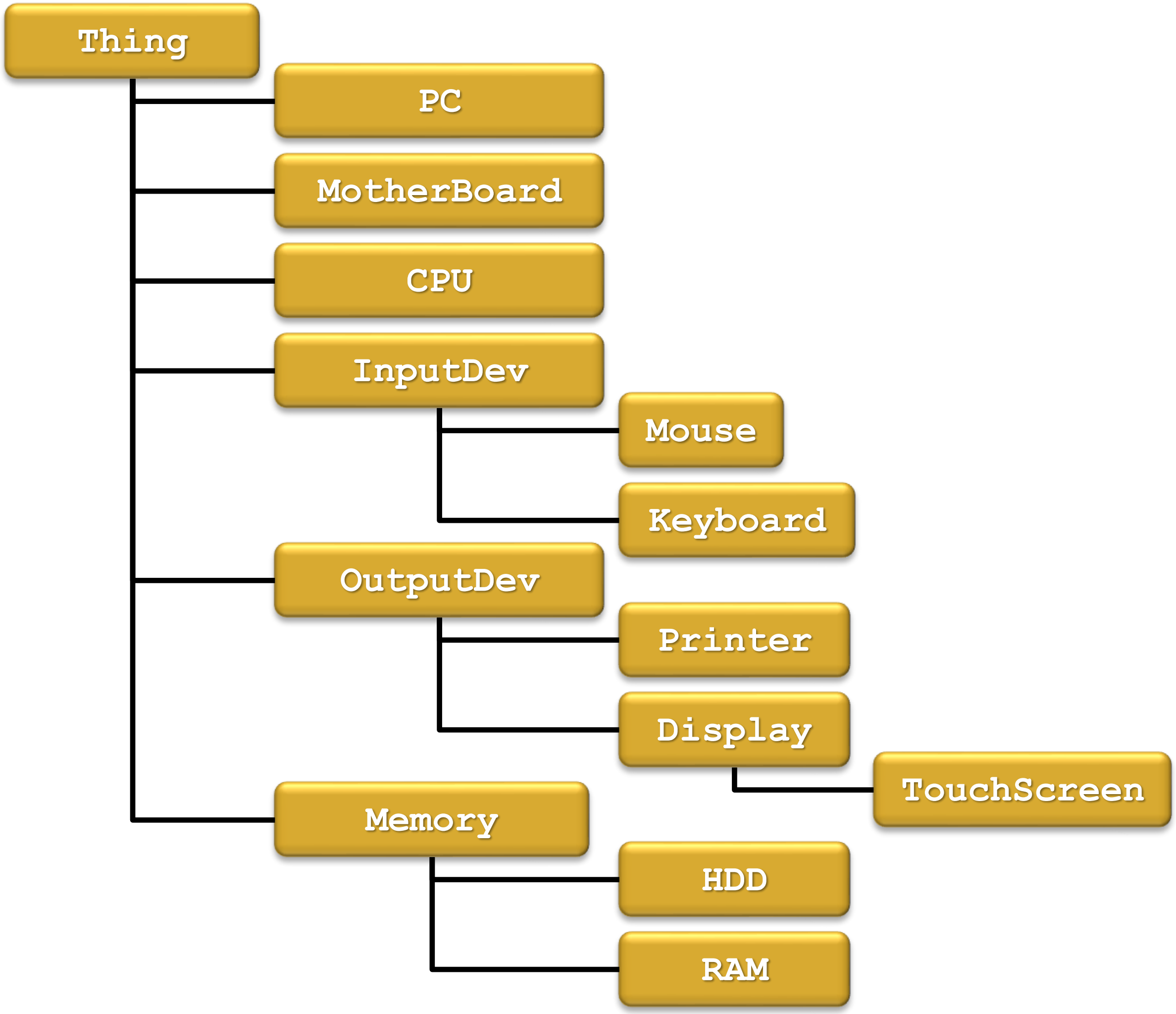
- **Instance** (individual)
  - Represents a concrete thing
  - Can belong to one or more classes
  - Example: johnDoe, appleGoldenDelicious, anger…

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ont: <http://www.john.com/myOntology.owl#> .

ont:benny rdf:type ont:Dog .
ont:superman a ont:ComicBookCharacter .
ont:mrBean <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ont:ComicCharacter .
```

Important parts of TBox

- Class hierarchy
  - Defines classes of things and their relationships (class-subclass and others)
- Object properties
  - Connections between two individuals
  - Example: *p:john p:loves p:mary* .
- Data properties
  - Connection between an individual and a value
  - Example: *p:john p:hasHeight "178.5"^^xsd:float .*

Sample ontology: class hierarchy



Properties

- In ontologies we define property's *domain* and *range*
  - ***Domain***: What can have this property
  - ***Range***: What can be the value of this property

```
med:hasDiagnosis
D: Human
R: Diagnosis
```

```
hum:isAttractedBy
D: Human
R: Human
```

```
phy:isAttractedBy
D: Particle
R: Particle
```

```
hum:hasSurname
D: Human
R: rdfs:Literal
```

```
psy:hasAtomicNumber
D: Atom
R: xsd:string
```

Properties

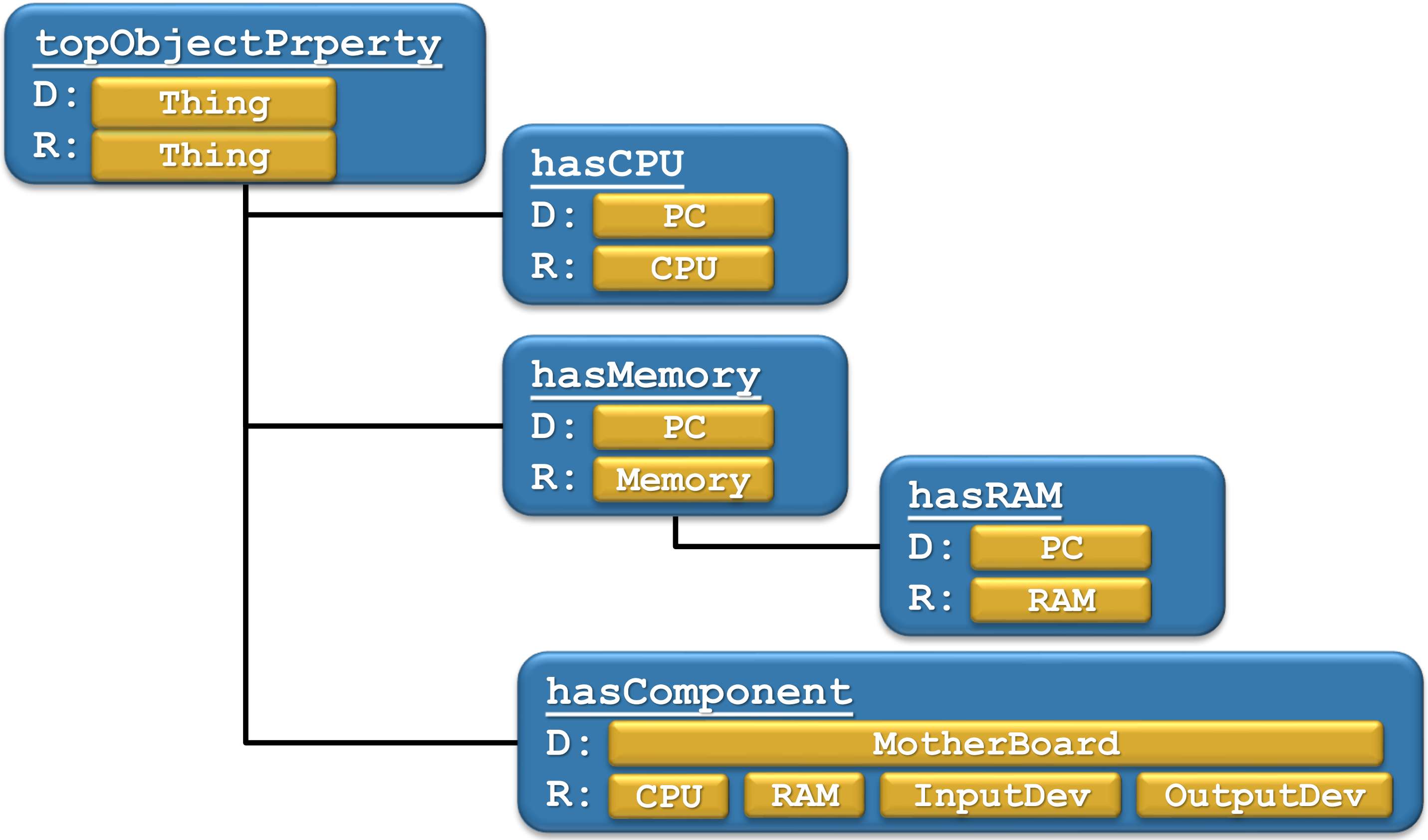
- Object properties:
  - Domain: *URI*
  - Range: *URI*

```
@prefix o: <http://john.com/myOnt.owl#> .

o:mary o:likes o:chocolate .
```

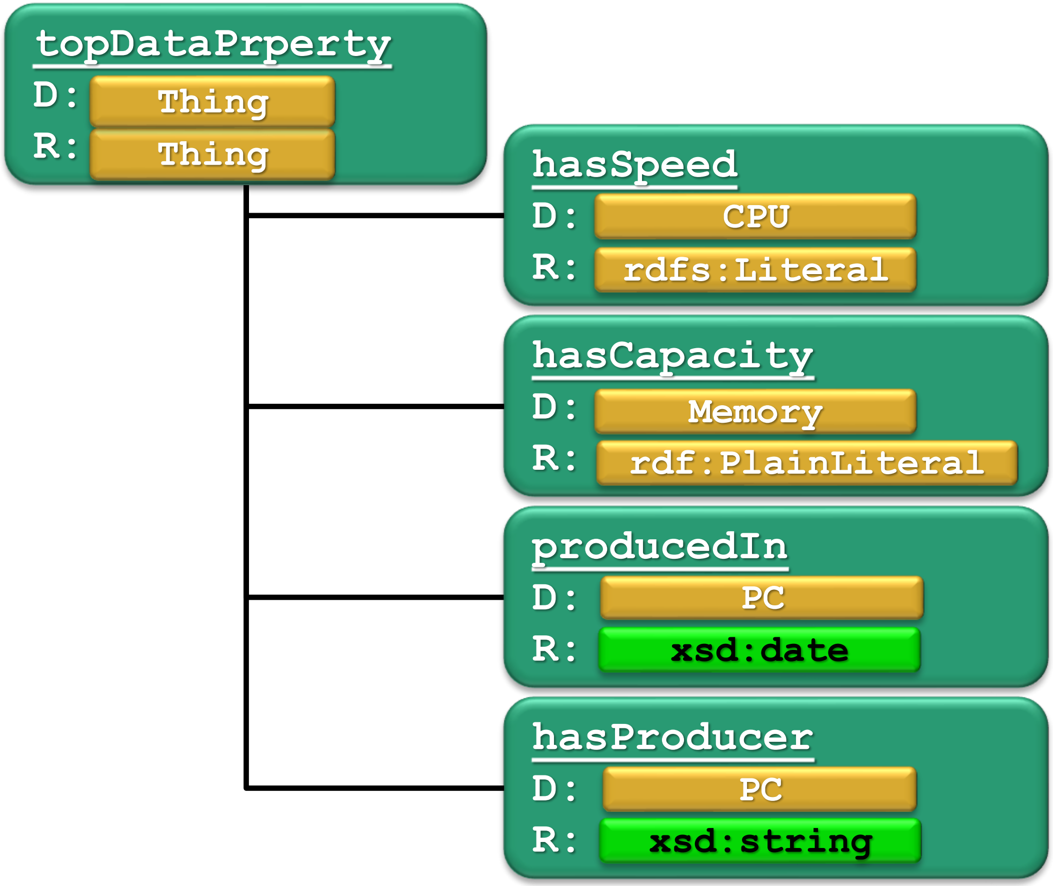
- Data properties:
  - Domain: *URI*
  - Range: *Literal* (typed or plain)

```
@prefix o: <http://john.com/myOnt.owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

o:mary o:age "30"^^xsd:int .
```

Sample ontology: object properties



Sample ontology: data properties



@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

Ontology language

- Language that is used to formally define ontologies
- Example:
  - ***RDFS*** (RDF Schema)
  - ***OWL*** (Web Ontology Language)
  - ***OWL2***
- Majority is based on RDF model as well
  - Ontology written in such language is RDF itself
- Differences between ontology languages
  - Expressiveness
  - Computational complexity of reasoning

RDF Schema (RDFS)

- Simple ontology language (W3C Recommendation in 2004)
- Prefix: `@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>`
- Features:
  - Declaration of classes and subclass hierarchy:

```
x:Human rdf:type rdfs:Class .
x:Human rdfs:subClassOf x:LivingBeing .
```

  - Declaration of literals and their hierarchy:

```
x:Henkilotunnus rdf:type rdfs:Literal .
rdfs:Datatype rdfs:subClassOf rdfs:Literal .
```

  - Definition of properties and their hierarchy:

```
x:hasAge rdf:type rdf:Property .
x:hasAge rdfs:domain x:LivingBeing .
x:hasAge rdfs:range xsd:int .
rdfs:subPropertyOf rdf:type rdf:Property .
x:hasMovablePart rdfs:subPropertyOf x:hasPart .
x:hasStaticPart rdfs:subPropertyOf x:hasPart .
```

  - Other features (*statement*, *container*, *collections*, *comments*, etc.)

RDFS example

- Ontology

```
@prefix x: <http://mypage.com/myOntologies/humanOntology#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

x:LivingBeing rdf:type        rdfs:Class .
x:Human       a               rdfs:Class ;
              rdfs:subClassOf x:LivingBeing .
x:hasAge      a               rdf:Property ;
              rdfs:domain     x:Human ;
              rdfs:range      xsd:int .
```

- Annotated resource

```
@prefix x: <http://mypage.com/myOntologies/humanOntology#> .
@prefix xsd: <http://www.w3.org/2000/01/rdf-schema#> .

x:bill a x:Human ; x:hasAge "40"^^xsd:int  .
```

```
@prefix x: <http://mypage.com/myOntologies/humanOntology#> .
@prefix xsd: <http://www.w3.org/2000/01/rdf-schema#> .

x:bill a x:LivingBeing ; x:hasAge  "40"^^xsd:int  .
```

OWL language

- **Web Ontology Language (OWL)** - is a semantic markup language for publishing and sharing ontologies on the World Wide Web.
- **OWL** is vocabulary extension RDF and derived from DAML+OIL Web Ontology Language.
- Two versions:
  - Version 1 (W3C Recommendation Feb 2004)
    - Dialects: OWL-Lite, OWL-DL, OWL-Full
  - Version 2 (W3C Recommendation Oct 2009)
    - Profiles: OWL EL, OWL QL, OWL RL
- Uses vocabulary from RDF and RDFS
- More expressive than RDFS

OWL version 1

- **OWL** has more expressive power than RDF Schema, provides additional vocabulary along with a formal semantics
- Three sublanguages:
  - **OWL Lite** was designed for easy implementation and to provide users with a functional subset that will get them started in the use of OWL.
  - **OWL DL** was designed to support the existing Description Logic business segment and to provide a language subset that has desirable computational properties for reasoning systems.
    - More expressive
    - Based on DL (Description Logic)
    - (Almost) all features included
    - Still computationally complete and decidable
  - **OWL Full** relaxes some of the constraints on OWL DL so as to make available features which may be of use to many database and knowledge representation systems, but which violate the constraints of Description Logic reasoners.
    - Maximum expressiveness
    - Computational properties not guaranteed

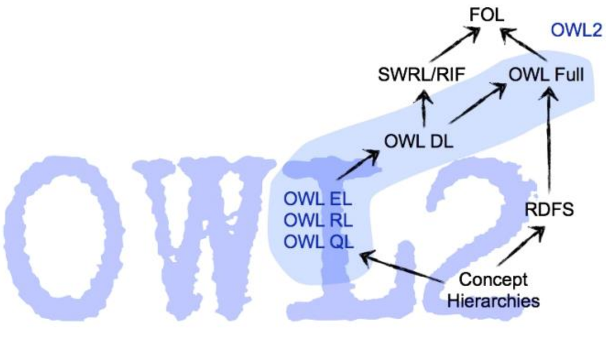
OWL version 2

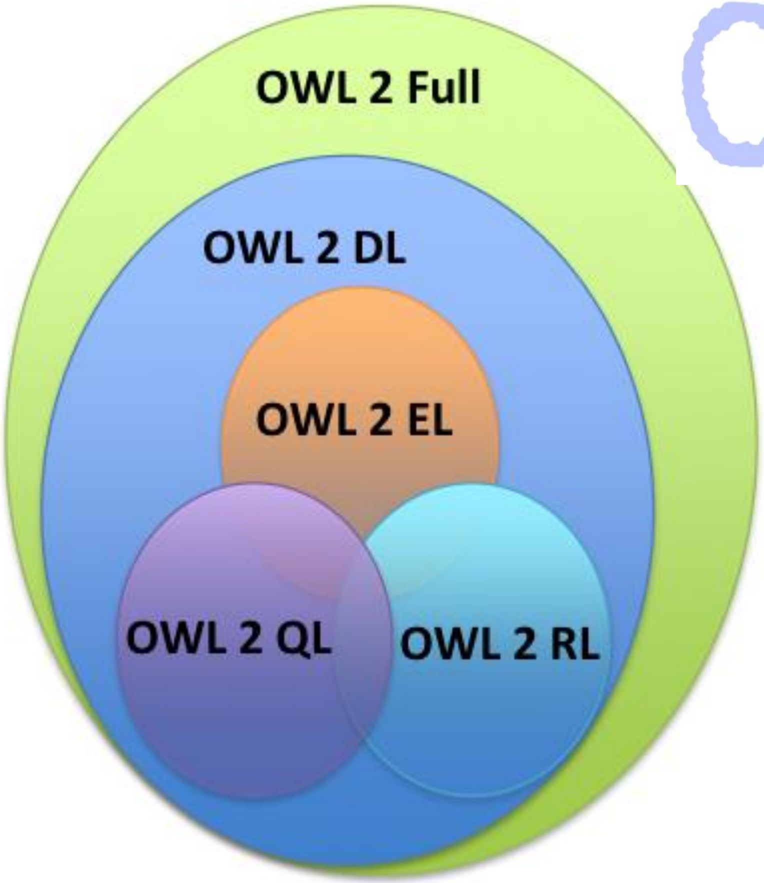
- **OWL 2** is extension of OWL designed to facilitate ontology development and sharing via the Web, with the ultimate goal of making Web content more accessible to machines.
  - OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents;
  - RDF/XML is primary exchange syntax for OWL 2 and provides interoperability of OWL 2 tools. Other alternative syntaxes also are used (Turtle, XML, Manchester Syntax, Functional-Style Syntax, etc.)
- **OWL 2 Profiles** (sublanguages) are syntactic restrictions of OWL 2. Each is more restrictive than OWL DL and provides different computational and/or implementational benefits:
  - **OWL 2 EL** enables polynomial time algorithms for all the standard reasoning tasks
    - applications with very large ontologies that need expressive power for performance
  - **OWL 2 QL** enables conjunctive queries to be answered in LogSpace using standard relational database technology
    - applications with relatively lightweight ontologies used to organize large numbers of individuals and need to access the data directly via relational queries (e.g., SQL)
  - **OWL 2 RL** enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples
    - applications with relatively lightweight ontologies used to organize large numbers of individuals and need to operate directly on data in the form of RDF triples

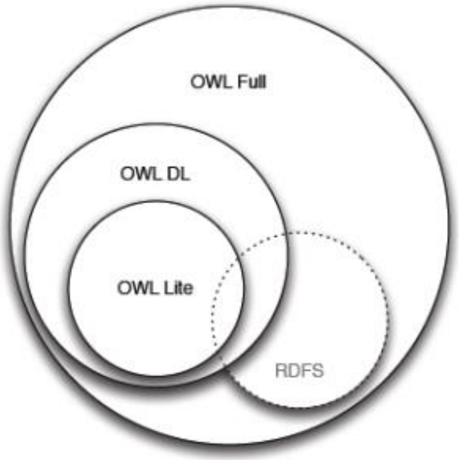
OWL version 2

- Additionally to three new profiles and new OWL 2 Manchester Syntax, **OWL 2** adds new functionality with respect to OWL 1:
  - syntactic sugar to make some common patterns easier to write (e.g., disjoint union of classes);
  - property chains and keys (in order to uniquely identify individuals of a given class by values of (a set of) key properties);
  - richer datatypes:
    - various kinds of ***numbers***: a wider range of XML Schema Datatypes (double, float, decimal, positiveInteger, etc.) and providing its own datatypes, e.g., owl:real;
    - ***strings*** with (or without) a Language Tag (using the rdf:PlainLiteral datatype);
    - boolean values, binary data, IRIs, time instants, etc.
  - datatype restrictions by means of constraining ***facets*** that constrain the range of values allowed for a given datataype, by length (for strings) *e.g., minLength, maxLength*, and minimum/maximum value, *e.g., minInclusive, maxInclusive*.
  - N-ary Datatypes;
  - qualified cardinality restrictions;
  - asymmetric, reflexive, and disjoint properties;
  - enhanced annotation capabilities.

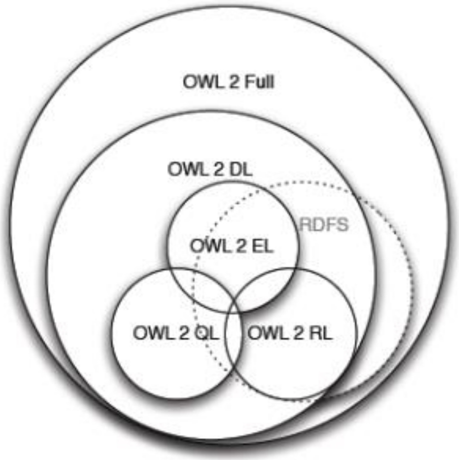
OWL







OWL 1



OWL 2 Profiles

Protégé

- **Protégé** is an ontology editor (*http://protege.stanford.edu/*)
- Documentation: *(http://protegewiki.stanford.edu/wiki/Main_Page)*
- Differences between Protege 3.x and 5.x (4.x) are equivalent to those between Frames based systems and OWL (and DL reasoning) based ones (http://users.jyu.fi/~olkhriye/ties4520/lectures/FramesAndOWLSideBySide.pdf)
  - Version 3.x (OWL 1 + RDFS)
  - Version 4.x & 5.x (OWL 2)
    - *written in a much more principled way than Protege 3 and for OWL ontologies Protege 5 (4) is generally the right choice;*
    - *does not include some of the plugins of Protege 3 and Protege 3 forms mechanism.*
- Many plugins
  - Reasoners (HermiT, Pellet, FaCT++)
  - Exporters
  - New views
- Manchester syntax
  - Used in Protégé to define set operations and property restrictions
  - More info: *http://www.w3.org/TR/owl2-manchester-syntax/*

WebProtege

- **WebProtege** is an open source, lightweight, web-based ontology editor (*http://protegewiki.stanford.edu/wiki/WebProtege*):
  - allows users to collaboratively develop ontologies in a distributed way;
  - supports OWL 2 ontologies;
  - users can upload OBO Format ontologies and edit them collaboratively
- WebProtege has a content management system.
  - Users can log in and upload their ontologies to the server, edit them, invite collaborators to contribute, and set permissions for collaborators (who can then view, edit, or make comments).
- Two modes of WebProtege:
  - **Local Mode**: WebProtégé loads the ontologies from a standalone instance of Protégé running in a servlet container (default mode);
  - **External Server Mode**: WebProtégé loads the ontologies from a Protégé server running outside of the servlet container, and acts as a web-based client connecting to the Protégé server.
- **WebProtege** On-line: *http://webprotege.stanford.edu/*

Part 2

Ontologies and Protégé

OWL document

- Parts
  - Ontology header
  - Class axioms
  - Property axioms
  - Facts about individuals
- Order of components is not important
- Extensions usually: `rdf`, `owl.` But supports many of main serializations…
- MIME type:
  - `application/rdf+xml` or
  - `application/xml`

OWL: Ontology header

- Ontology is a resource as well, therefore can have own annotations (properties).

- Annotations (**owl:AnnotationProperty**):
  - owl:versionInfo - string that provides version information (does not influence the logical meaning of the ontology);
  - owl:priorVersion - identifies the ontology as a prior version of the containing ontology;
  - owl:backwardCompatibleWith - identifies the specified ontology as a prior version of the containing ontology, and further indicates that it is backward compatible with it;
  - owl:incompatibleWith - indicates that the containing ontology is a later version of the referenced ontology, but is not backward compatible with it.
  - also: rdfs:label, rdfs:comment, rdfs:seeAlso, rdfs:isDefinedBy
  - also ***(OWL-2)***: owl:deprecated used to specify whether IRI is deprecated or not
- Ontology imports (**owl:imports**)
  - Imports another ontology that is considered to be a part of the importing ontology

OWL: Ontology header

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <http://jyu.fi/ontology1.owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

<http://jyu.fi/ontology1.owl> rdf:type owl:Ontology ;
   rdfs:comment "simple family ontology"@en ;
   owl:backwardCompatibleWith <http://jyu.fi/ontology0.owl> .
```

OWL: Class axioms

- Class descriptions
  1. Plain declaration – a class identifier (URI reference)
  2. Exhaustive enumeration of all individuals
  3. Property restriction
  4. Set operations:
     - Intersection of classes;
     - Union of classes;
     - Complement of a class.
- Each class belongs to ***owl:Class***
  - *owl:Class* is a subclass of *rdfs:Class*
- Special classes:
  - *owl:Thing* (class with all individuals);
  - *owl:Nothing* (class with no individuals, empty set).

owl:Thing and owl:Nothing

- **owl:Thing**
  - Contains all the individuals in the world
  - Automatically parent of every other class
  - Any individual is automatically a member of this class
  - Any class is automatically a subclass of *owl:Thing*
- **owl:Nothing**
  - No individual belongs to this class (empty set)
  - Automatically subclass of all other classes
    - Empty set is always a subset of any non-empty set
  - Automatically disjoint with other classes
    - Empty set is always disjoint with any non-empty set

Classes: Axioms

- **Axiom**
  - Formula in a formal language that is universally valid and describes knowledge that cannot be expressed simply with the help of other existing components.
  - Some statement ("rule") that is always true
  - It is given, you don't question it or prove it
- **Necessary condition**
  - *X* is necessary condition for *Y: (Y=>X)*
  - Example: Having PhD. is a necessary condition for being a professor (but not sufficient)
- **Sufficient condition**
  - *X* is sufficient condition for *Y: (X=>Y)*
  - Stronger than necessary condition
  - Example: Being a human is a sufficient condition for being a living being (but not necessary)

Classes: Axioms 2

- Class-subclass axiom
  - **rdfs:subClassOf** (came from RDFS)
  - Same meaning as in RDFS
- Equivalence axiom (**owl:equivalentClass**)
  - Class description has exactly the same meaning as some other class description (they represent the same set)
- Disjointness axiom (**owl:disjointWith**) (*is not part of OWL Lite*)
  - Only necessary condition, not sufficient
  - You specify what the class ***is not*** about
  - You do not specify what the class ***is*** about
  - Example: *Car is disjoint with Bicycle*
  - *a shortcut to define several classes to be disjunctive*

```
_:x45 rdf:type owl:AllDisjointClasses;
        owl:members (:Car :Human :Organization).
```

Classes: 1.Plain declaration, 2.Enumeration

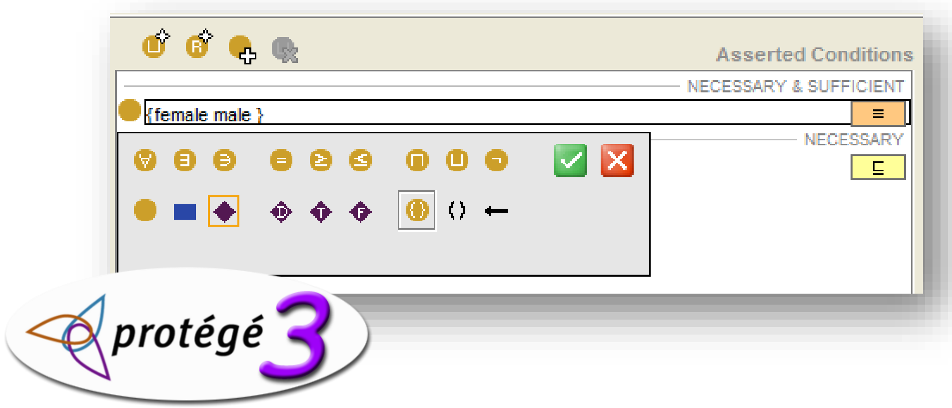
- Plain declaration
  - You specify that some URI represents a class

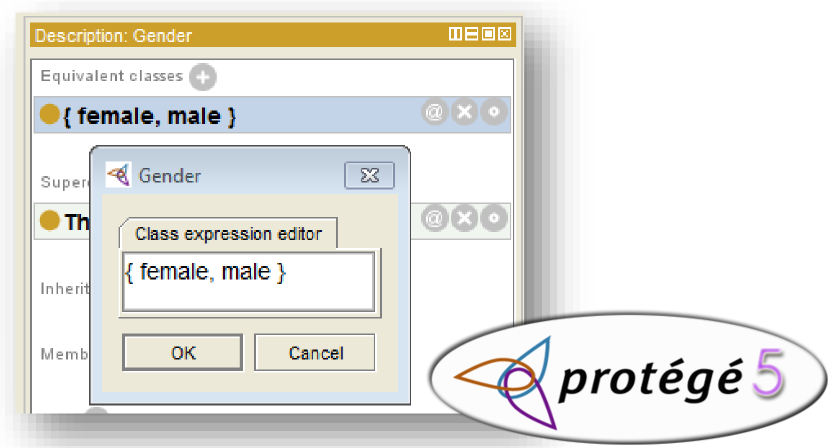
```
ex:Human rdf:type owl:Class
```

- Enumeration (*is not part of OWL Lite*)
  - You define the Class by saying what individuals belong to it. The Class has exactly those individuals, nothing more, nothing less;
  - Use *owl:oneOf* predicate. Value must be a list of individual of that class;

```
ex:Gender owl:oneOf (ex:female ex:male)
```

  - Example: Continent, Gender, Grade, etc.





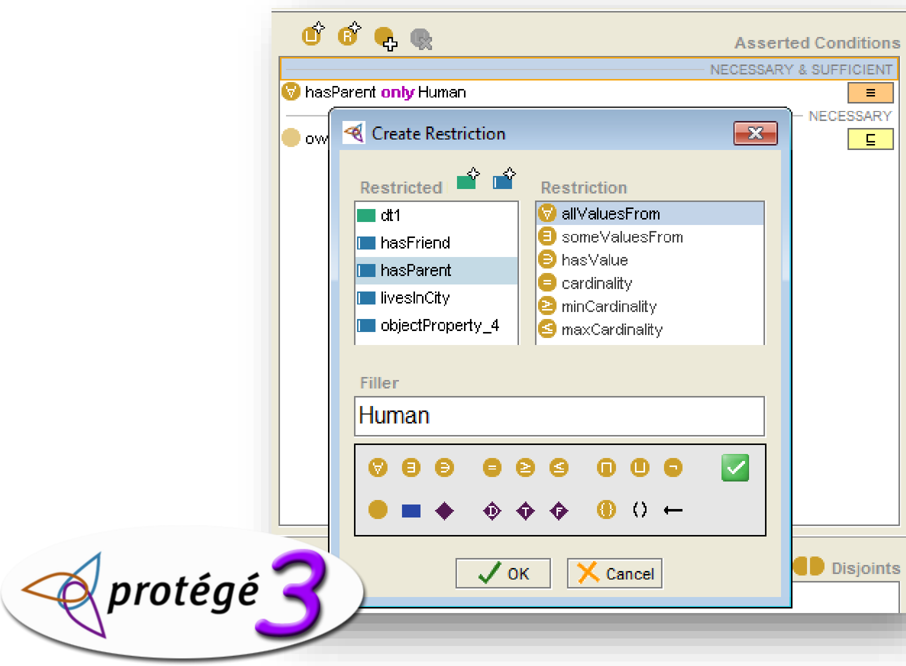
Classes: 3.Property restrictions

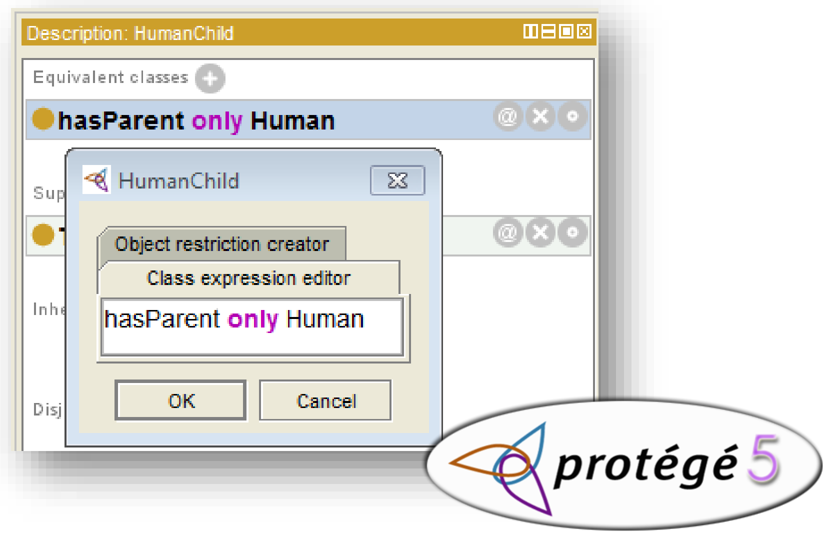
- Anonymous class (restriction) defined by specifying restrictions on its properties
- **owl:Restriction** is a subclass of *owl:Class*
- Restrictions:
  - Value constraint:
    - *owl:allValuesFrom*,
    - *owl:someValuesFrom*,
    - *owl:hasValue* (*is not part of OWL Lite*)
  - Cardinality constraint:
    - *owl:cardinality* (***OWL Lite*** supports cardinality constraint with only values “0” or ”1”),
    - *owl:minCardinality* and *owl:maxCardinality*,
    - *owl:qualifiedCardinality* ***(OWL-2),***
    - *owl:minQualifiedCardinality* and *owl:maxQualifiedCardinality* ***(OWL-2)***
  - Self-Restriction:
    - *owl:hasSelf* ***(OWL-2)***

Classes: 3.Property restrictions

- Value constraint: ***owl:allValuesFrom***

```
:HumanChild rdf:type owl:Class ;
            owl:equivalentClass [
                     rdf:type owl:Restriction ;
                     owl:onProperty :hasParent ;
                     owl:allValuesFrom :Human
             ] .
```

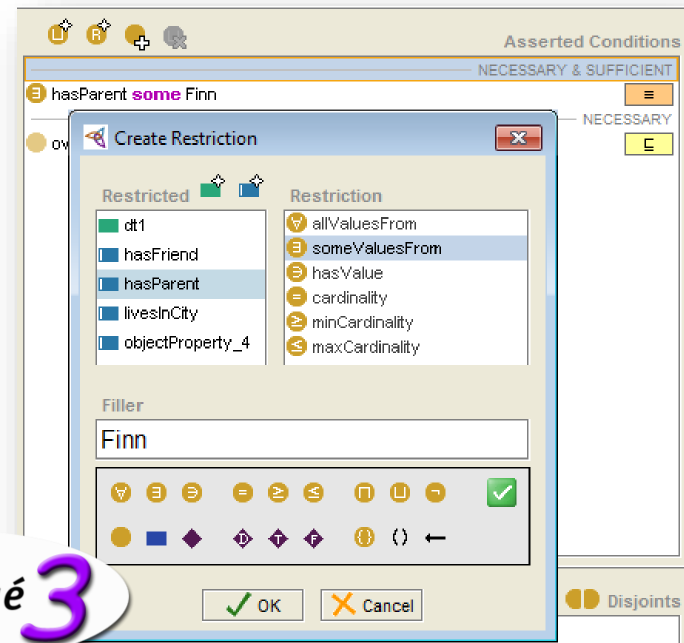


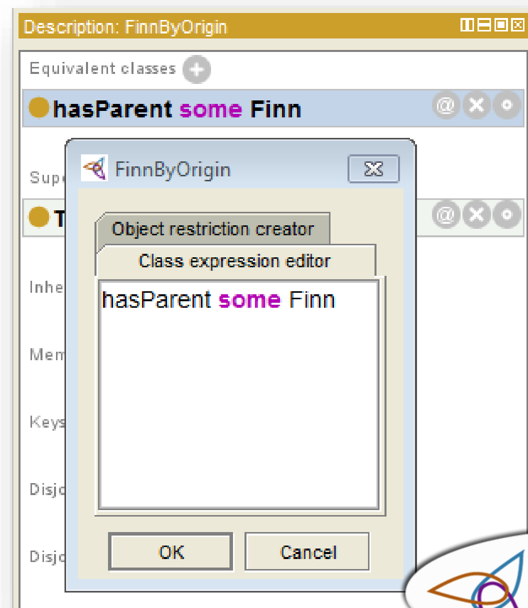


Classes: 3.Property restrictions

- Value constraint: ***owl:someValuesFrom***

```
:FinnByOrigin rdf:type owl:Class ;
              owl:equivalentClass [
                       rdf:type owl:Restriction ;
                       owl:onProperty :hasParent ;
                       owl:someValuesFrom :Finn
              ] .
```

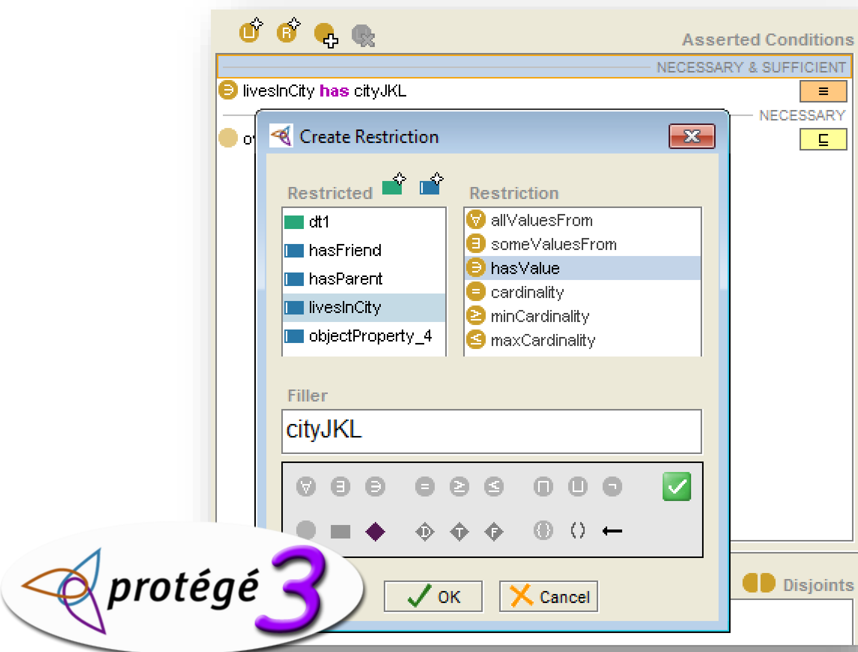


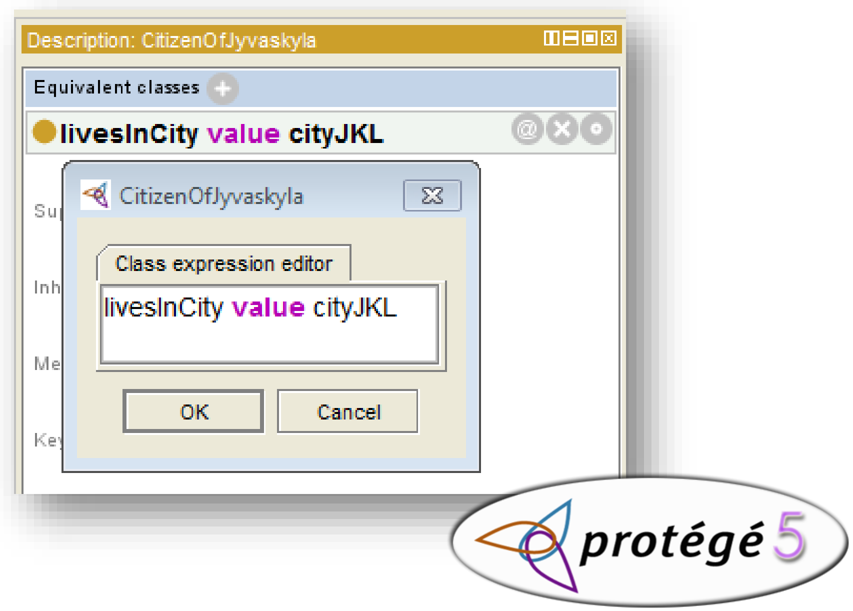


Classes: 3.Property restrictions

- Value constraint: ***owl:hasValue***

```
:CitizenOfJyvaskyla rdf:type owl:Class ;
                    owl:equivalentClass [
                              rdf:type owl:Restriction ;
                              owl:onProperty :livesInCity ;
                              owl:hasValue :cityJKL
                    ] .
```





Classes: 3.Property restrictions

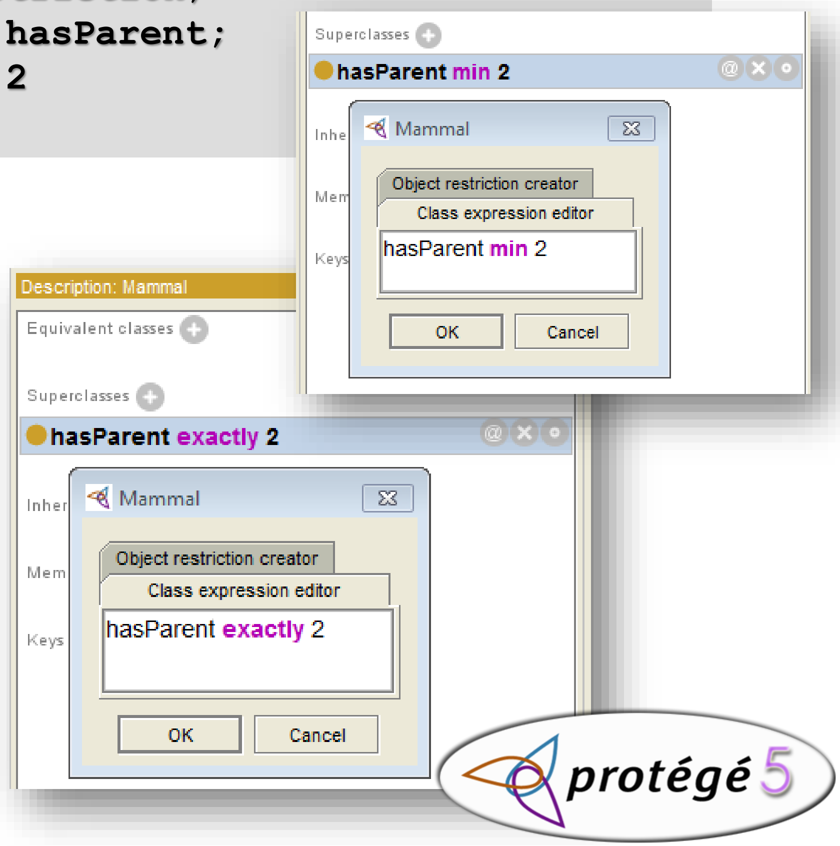
- Cardinality constraint example:

```
:Mammal rdf:type owl:Class;
        rdfs:subClassOf [
                rdf:type   owl:Restriction;
                owl:onProperty    :hasParent;
                owl:cardinality    2
        ];
        rdfs:subClassOf [
                rdf:type   owl:Restriction;
                owl:qualifiedCardinality   1;
                owl:onProperty    :hasParent;
                owl:onClass :Female
        ];
        rdfs:subClassOf [
                rdf:type   owl:Restriction;
                owl:qualifiedCardinality   1;
                owl:onProperty    :hasParent;
                owl:onClass :Male
        ].
```

Classes: 3.Property restrictions

- Cardinality: ***owl:cardinality***, ***owl:minCardinality***, ***owl:maxCardinality***

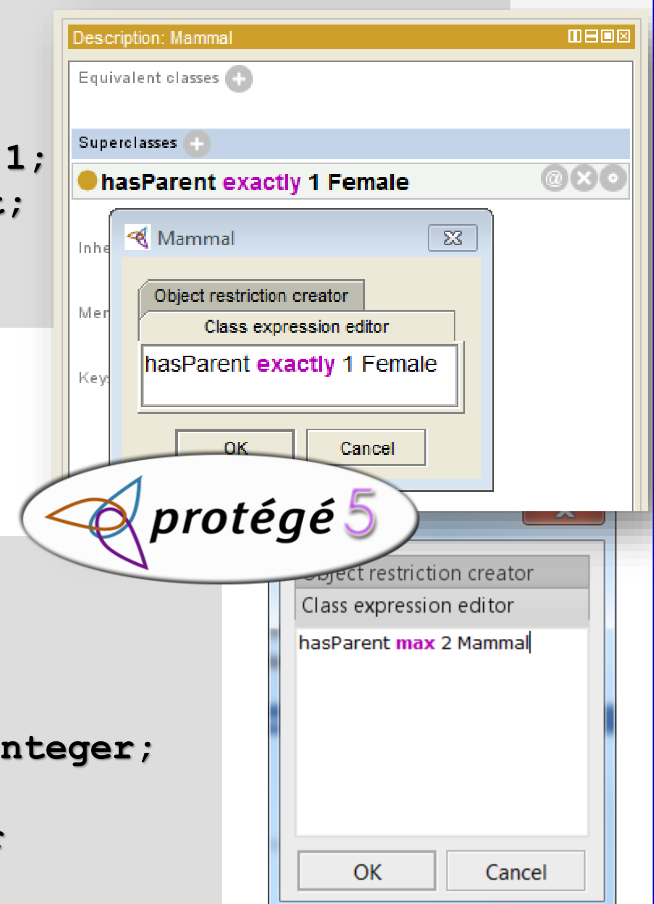
```
:Mammal rdf:type owl:Class;
        rdfs:subClassOf [
                rdf:type   owl:Restriction;
                owl:onProperty    :hasParent;
                owl:cardinality    2
        ]…
```



Classes: 3.Property restrictions

- Qualified cardinality: ***owl:qualifiedCardinality***, ***owl:minQualifiedCardinality***, ***owl:maxQualifiedCardinality*** ***(OWL-2)***

```
:Mammal rdf:type owl:Class;
        rdfs:subClassOf [
                rdf:type   owl:Restriction;
                owl:qualifiedCardinality   1;
                owl:onProperty   :hasParent;
                owl:onClass :Female
        ]…
```



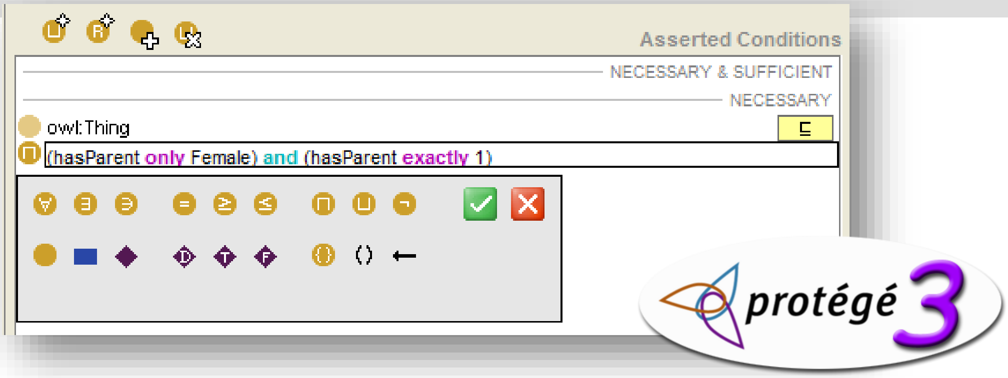
- Also can be used with Datatype properties

```
:Person rdf:type owl:Class;
        rdfs:subClassOf [
                rdf:type   owl:Restriction;
                owl:qualifiedCardinality
                      "1"^^xsd:nonNegativeInteger;
                owl:onProperty   :hasAge;
                owl:onDataRange xsd:integer
        ]…
```

Classes: 3.Property restrictions

- Qualified cardinality *is not a part of OWL-1*. Such restriction can be done via intersection of two other restrictions:

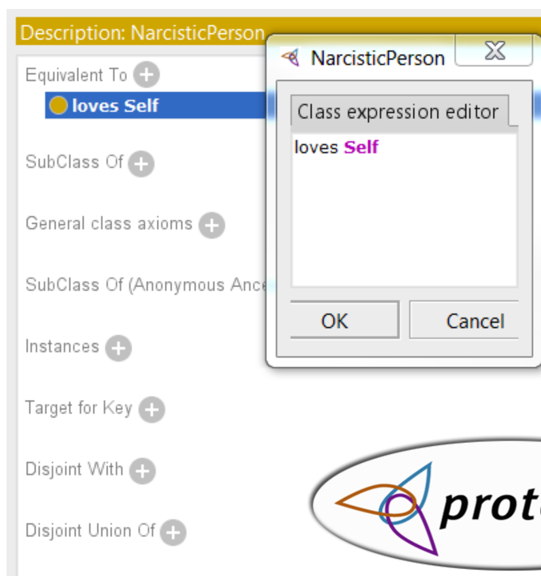
```
:Mammal rdf:type owl:Class;
    rdfs:subClassOf [ rdf:type owl:Class ;
                      owl:intersectionOf (
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasParent ;
                              owl:allValuesFrom :Female ]
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasParent ;
                              owl:cardinality "1"^^xsd:int ; ]
                      )
    ] …
```



Classes: 3.Property restrictions

- Self-restriction: ***owl:hasSelf*** ***(OWL-2)***

```
:NarcisticPerson rdf:type owl:Class ;
                         owl:equivalentClass [
                                    rdf:type owl:Restriction ;
                                    owl:onProperty :loves ;
                                    owl:hasSelf "true"^^xsd:boolean
                         ] .
```



Classes: 3.Property restrictions

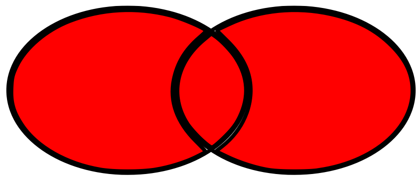
- Constraining facets can be used to *restrict datatype values*. The following constraining facets can be used:
  - Numbers and time instants: *xsd:minInclusive*, *xsd:maxInclusive*, *xsd:minExclusive*, *xsd:maxExclusive*
  - Strings and IRIs: *xsd:minLength*, *xsd:maxLength*, *xsd:length*, *xsd:pattern*
  - Binary data: *xsd:minLength*, *xsd:maxLength*, *xsd:length*

  Example: class ***Teenager*** is defined as those who are between 13 and 19 years old (both inclusive).

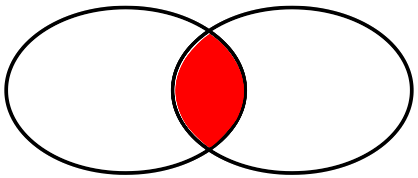
```
:Teenager rdfs:subClassOf _:x .
_:x rdf:type owl:Restriction ;
    owl:onProperty  :hasAge ;
    owl:someValuesFrom _:y .
_:y rdf:type rdfs:Datatype ;
    owl:onDatatype xsd:integer ;
    owl:withRestrictions ( _:z1 _:z2 ) .
_:z1 xsd:minInclusive "13"^^xsd:integer .
_:z2 xsd:maxInclusive "19"^^xsd:integer .
```

Classes: 4. Set operations

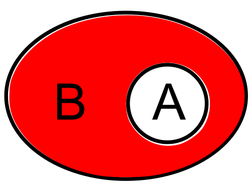
Set theory basics



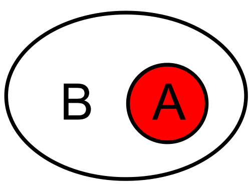
Union (A or B)



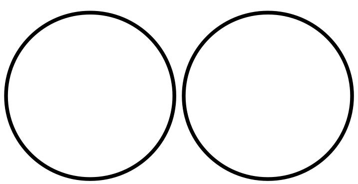
Intersection (A and B)



Complement
(complement of A inside B)



Set-subset
(A is subset of B)

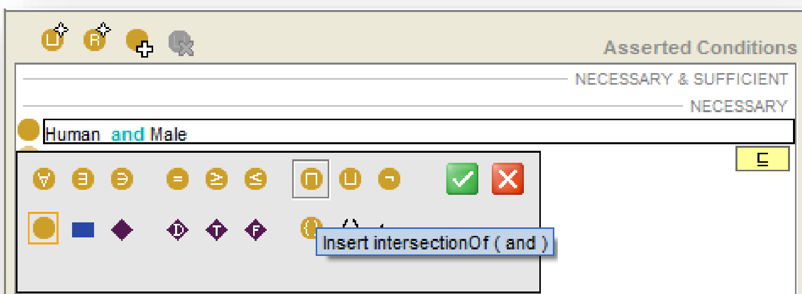


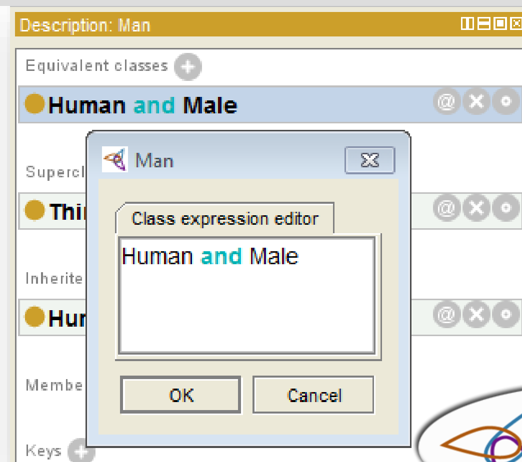
Disjoint sets

Classes: 4. Set operations

- Intersection (*has some restrictions in OWL Lite*)
  - ***owl:intersectionOf*** (= logical AND)
  - Example: class ***Man*** is intersection of classes ***Male*** and ***Human***
  - Example: ***Man*** = ***Male*** AND ***Human***

```
:Man rdf:type owl:Class ;
     owl:equivalentClass [
                 rdf:type owl:Class ;
                 owl:intersectionOf ( :Human :Male)
     ] ;
```





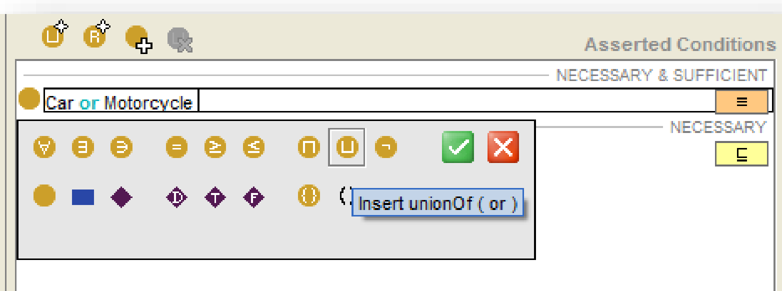
protégé 3

protégé 5

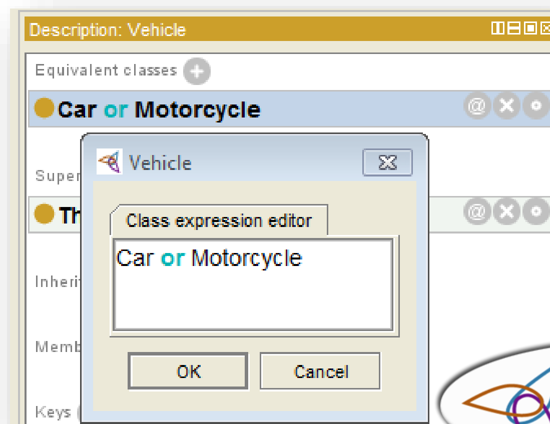
Classes: 4. Set operations

- Union (*is not part of OWL Lite*)
  - ***owl:unionOf*** (= logical OR)
  - Example: class ***Vehicle*** is union of classes ***Car*** and ***Motorcycle***
  - Example: ***Vehicle*** = ***Car*** OR ***Motorcycle***

```
:Vehicle rdf:type owl:Class ;
          owl:equivalentClass [
                   rdf:type owl:Class ;
                   owl:unionOf (:Car :Motorcycle )
          ] ;
```



protégé 3

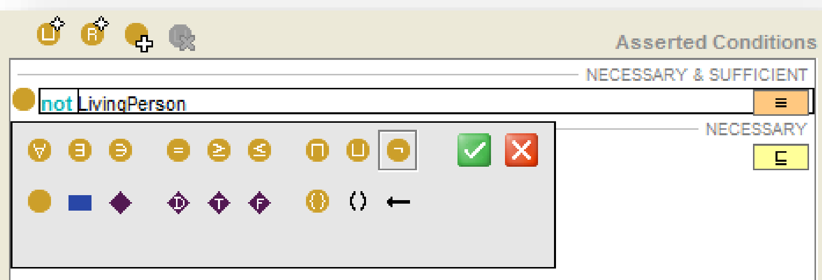


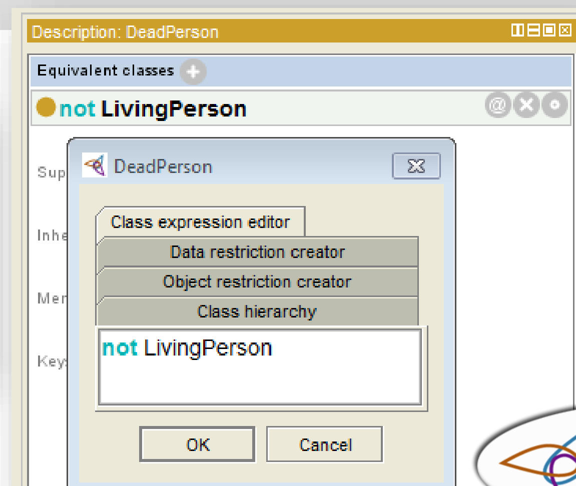
protégé 5

Classes: 4. Set operations

- Complement (*is not part of OWL Lite*)
  - ***owl:complementOf*** (logical NOT)
  - Example: class ***DeadPerson*** is complement of class ***LivingPerson***
  - Example: ***DeadPerson*** = NOT ***LivingPerson***

```
:DeadPerson rdf:type owl:Class ;
            owl:equivalentClass [
                      rdf:type owl:Class ;
                      owl:complementOf :LivingPerson
            ] ;
```

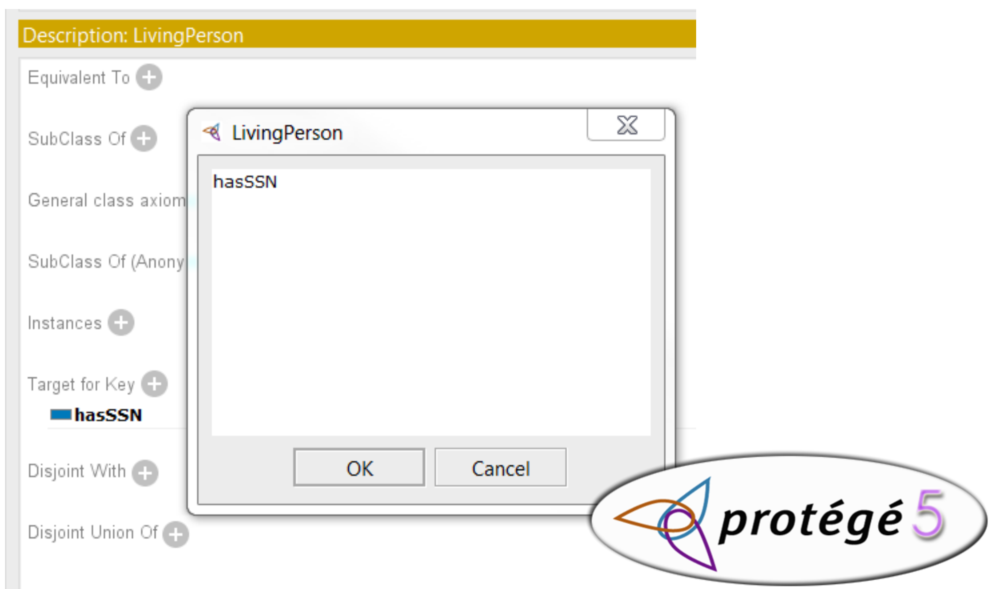




Classes

- Keys: ***owl:hasKey*** ***(OWL-2)***
  - a collection of (data or object) properties can be assigned as a key to a class expression. This means that each named instance of the class expression is uniquely identified by the set of values which these properties attain in relation to the instance.
  - Example: the identification of a person by his/her social security number (SSN)

```
:Person owl:hasKey ( :hasSSN ) .
```



Properties:

- In OWL we already recognize 2 properties
  - **Object property** (class *owl:ObjectProperty*)
  - **Datatype property** (class *owl:DatatypeProperty*)
  - There are two other type of properties that are used in OWL DL (*owl:AnnotationProperty* and *owl:OntologyProperty* classes).
- All of them are subclass of *rdf:Property*
- Special properties:
  - *owl:topObjectProperty* (the object property that relates every two individuals).
  - *owl:topDataProperty* (the data property that relates every individual to every data value).
- More property axioms:
  - Old RDFS: *rdfs:subPropertyOf*, *rdfs:domain*, *rdfs:range*
  - Relation to other properties: *owl:equivalentProperty*, *owl:inverseOf*
  - Global cardinality constraints: *owl:FunctionalProperty*, *owl:InverseFunctionalProperty*
  - Logical characteristics: *owl:SymmetricProperty*, *owl:TransitiveProperty*
  - Logical characteristics ***(OWL-2)***: *owl:AsymmetricProperty*, *owl:ReflexiveProperty*, *owl:IrreflexiveProperty*
  - Property chains ***(OWL-2):*** *owl:propertyChainAxiom*

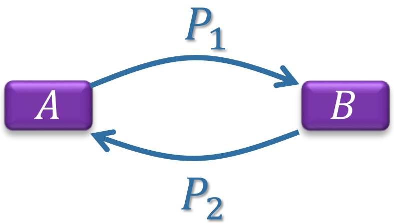
Old RDFS axioms

- ***rdfs:subPropertyOf***
  - Same meaning as in RDFS (sublanguage limitations must be taken into account)
- ***rdfs:domain*** and ***rdfs:range***
  - Same meaning as in RDFS;
  - Multiple axioms allowed and interpreted as a conjunction (intersection of provided classes);
  - If union of classes is needed, then use *owl:unionOf*

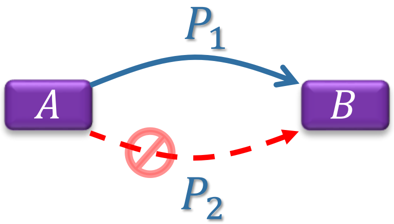
```
:hasFriend rdf:type owl:ObjectProperty ;
   rdfs:domain :Human ;
   rdfs:range [ rdf:type owl:Class ;
                owl:unionOf ( :Animal :Human )
              ] .
```

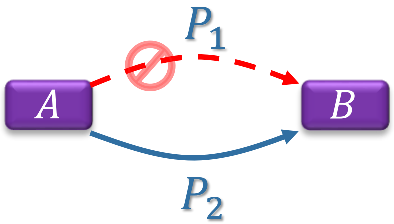
Relation to other properties

- *owl:equivalentProperty*
  - Equivalence of two properties.
- *owl:inverseOf*
  - Simply: property P_1 is inverse property of the property P_2, if range and domain of these properties are switched (direction of "arrow" is switched);
  - Example: properties *ex:isOwnedBy* & *ex:owns* are inverse, *ex:hasChild* & *ex:hasParent* are inverse.



- *owl:propertyDisjointWith* ***(OWL-2)***
  - Simply: properties P_1 and P_2 are disjunctive, if two individuals are never related via both properties;
  - Example: properties *ex:hasParent* & *ex:hasChild* are disjunctive.
  - *a shortcut to define several properties to be disjunctive*

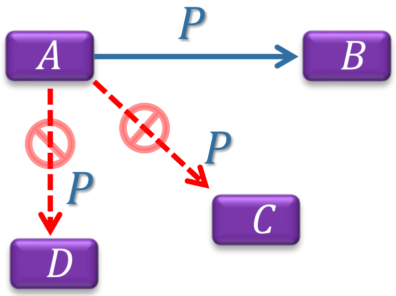




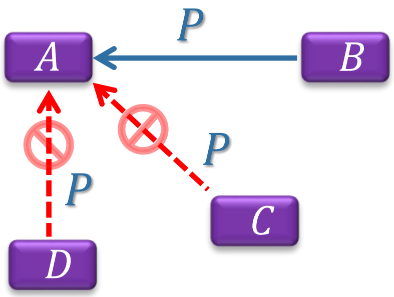

```
_:x25 rdf:type owl:AllDisjointProperties;
      owl:members (:hasParent :hasChild :hasGrantchild).
```

Global cardinality constraints

- *owl:FunctionalProperty*
  - Simply: such property can have only one value. Property may relate individual A only to one individual;
  - Example: *ex:marriedTo* (in monogamous cultures);

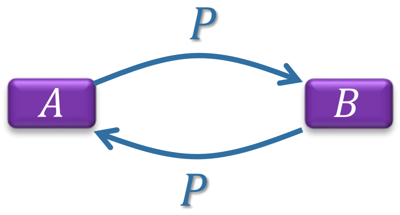


- *owl:InverseFunctionalProperty*
  - Simply: such property cannot relate two or more individuals (only one) to the same destination individual A;
  - Example: *ex:biologicalMotherOf*

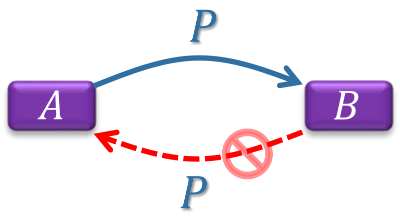


Logical characteristics

- *owl:SymmetricProperty*
  - Simply: if property P relates individual A to individual B, then the same property P also relates individual B to individual A;
  - Example: *ex:hasSpouse.*

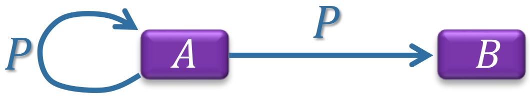


- *owl:AsymmetricProperty* **(OWL-2)**
  - Simply: If the property P relates individual A to individual B, then individual B cannot be related to individual A via the same property P;
  - Example: *ex:isChildOf.*

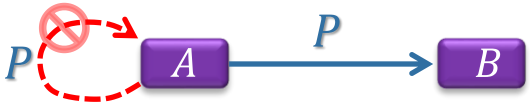


Logical characteristics (OWL-2)

- *owl:ReflexiveProperty* ***(OWL-2)***
  - Simply: If the property P relates individual A to individual A (to itself) and at the same time the property P may relate individual A to other individuals;
  - Example: *ex:knows.*

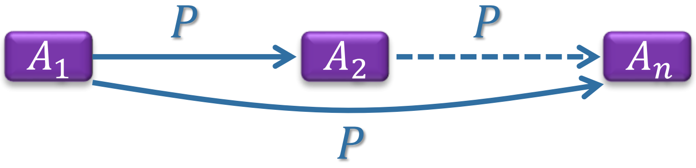


- *owl:IrreflexiveProperty* ***(OWL-2)***
  - Simply: If the property P relates individual A to individual B, then individuals A and B are not the same individuals;
  - Example: *ex:motherOf.*



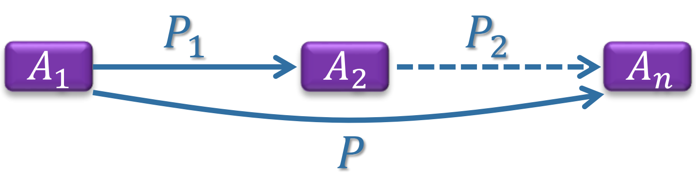
Logical characteristics

- *owl:TransitiveProperty*
  - Simply: if property P relates individual A_1 to individual A_2 , and the same property P relates individual A_2 to individual A_n, then the same property P also relates individual A_1 to individual A_n;
  - Example: *ex:bossOf, ex:hasAncestor.*



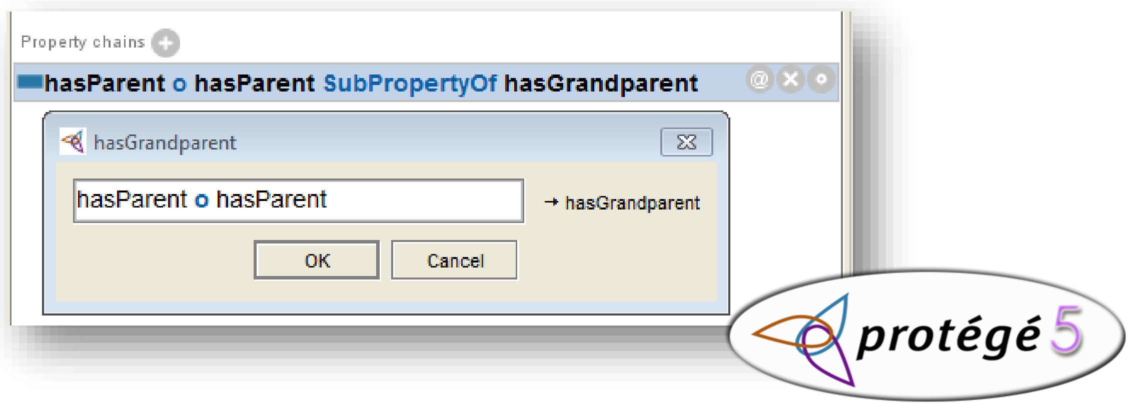
Property chains (OWL-2)

- *owl:propertyChainAxiom* ***(OWL-2)***
  - Simply: If the property P_1 relates individual A_1 to individual A_2, and property P_2 relates individual A_2 to individual A_n, then property P relates individual A_1 to individual A_n;



  - Example:

```
:hasGrandparent rdf:type owl:ObjectProperty ;
    owl:propertyChainAxiom ( :hasParent :hasParent ) .

:hasComponentFrom rdf:type owl:ObjectProperty ;
    owl:propertyChainAxiom ( :hasComponent :hasCountryOfOrigin ) .
```



Individuals' identity

- Equality:
  - Predicate ***owl:sameAs***;
  - Saying that URI1 and URI2 mean the same individual.
- Non-equality:
  - Predicate ***owl:differentFrom***;
  - Saying that URI1 and URI2 are definitely not the same individual.
- Different among each other:
  - property ***owl:distinctMembers*** is defined as a predicate that links an instance of ***owl:AllDifferent*** class to a list of individuals which are all different from each other;
  - Saying that URI1, …, URIn are all different from each other.

```
_:x39 rdf:type owl:AllDifferent;
      owl:distinctMembers (f:John f:Mary f:Bill f:Susan).
```

- *Important:* *If no information about equality or non-equality is specified, then we must assume that both are possible.*

Individuals' identity

- *Negated Property Instantiation*:
  - Two individuals can be explicitly defined as "*not related to each other*" via a given property

```
_:x29 rdf:type owl:NegativePropertyAssertion ;
      owl:sourceIndividual :Bob ;
      owl:assertionProperty :isBrother ;
      owl:targetIndividual :Michael .
```

  - Can also be used with Datatype properties…

```
_:x19 rdf:type owl:NegativePropertyAssertion ;
      owl:sourceIndividual :Bob ;
      owl:assertionProperty :hasAge ;
      owl:targetValue "53"^^xsd:integer.
```

OWL Full

- All the constructs are allowed;
- ***owl:Class*** is equivalent to ***rdfs:Class***;
- ***owl:Thing*** is equivalent to ***rdfs:Resource***;
- ***owl:ObjectProperty*** is equivalent to ***rdf:Property***. Therefore, ***datatype property*** is subclass of ***object property***;
- Very expressive (a lot of "freedom" to define things);
- You lose some guarantees on computability.

OWL DL

- Requires disjointness of:
  - classes, properties (datatype properties, object properties, annotation properties, ontology properties), individuals, data values, datatypes, built-in vocabulary
  - This has many implications…
- All axioms must be:
  - well-formed
  - with no missing or extra components
  - must form a tree-like structure

```
:Car rdf:type owl:Class ;
     rdfs:subClassOf :Vehicle .
```

… not enough

```
:Vehicle rdf:type owl:Class .
```

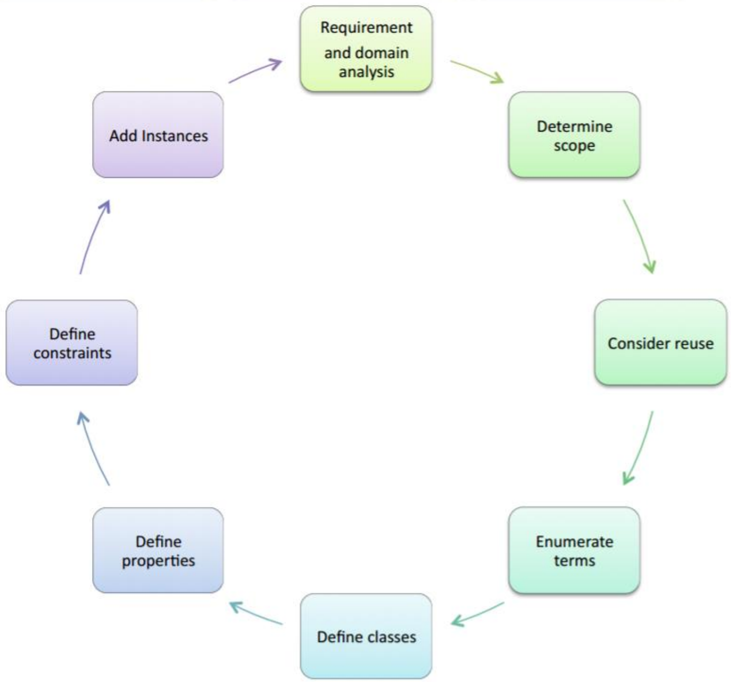
OWL Lite

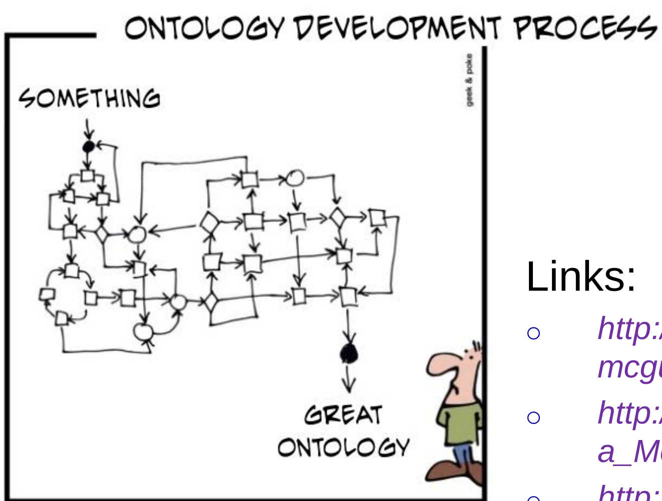
- Least expressive
- “Minimal useful subset of language features, that are relatively straightforward for tool developers to support”
- No use of:
  - *owl:oneOf*
  - *owl:unionOf*
  - *owl:complementOf*
  - *owl:hasValue*
  - *owl:disjointWith*
  - *owl:DataRange*
- + some other limitations

Ontology Development Process

In theory:

In reality: *it is an iterative process…*





acc. to http://geekandpoke.typepad.com/geekandpoke/2012/01/simply-explained-dp.html

Links:

- *http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html*
- *http://smartdata2015.dataversity.net/uploads/handouts/TUE_0830_Kendall_Elisa_McGuinness_Deborah_COLOR_7736.pdf*
- *http://slideplayer.com/slide/4414078/*

Further reading

- OWL Reference guide
  - *http://www.w3.org/TR/owl-ref/*
  - Easy to understand, many examples
  - Good chapters
    - All language elements (*http://www.w3.org/TR/owl-ref/#appA*)
    - Differences between sublanguages (*http://www.w3.org/TR/owl-ref/#Sublanguage-def*)
    - Tips (*http://www.w3.org/TR/owl-ref/#app-DLinRDF*)
- OWL-2 (*http://www.w3.org/TR/owl2-syntax/* and *http://www.w3.org/TR/owl2-primer/*)
- Ontology editors (*https://www.w3.org/wiki/Ontology_editors*)
  - TopBraid Composer;
  - Fluent Editor;
  - Knoodl;
  - Semantic Turkey;
  - NeOn Toolkit;
  - Etc.

Task 3