

# Lecture Notes

# Big Data in Medical Informatics

## Week 4:

## Ontologies Part 2

Oya Beyan, Ph.D.  
Prof. Dr. Stefan Decker

# Protégé OWL Overview



## Classes

- Subclass relationships
- Disjoint classes



## Properties

- Characteristics (transitive, inverse)
- Range and Domain



ObjectProperties (references)



DatatypeProperties (simple values)



## Individuals

- Property values



## Class Descriptions

- Restrictions
- Logical expressions

OWL for data  
exchange

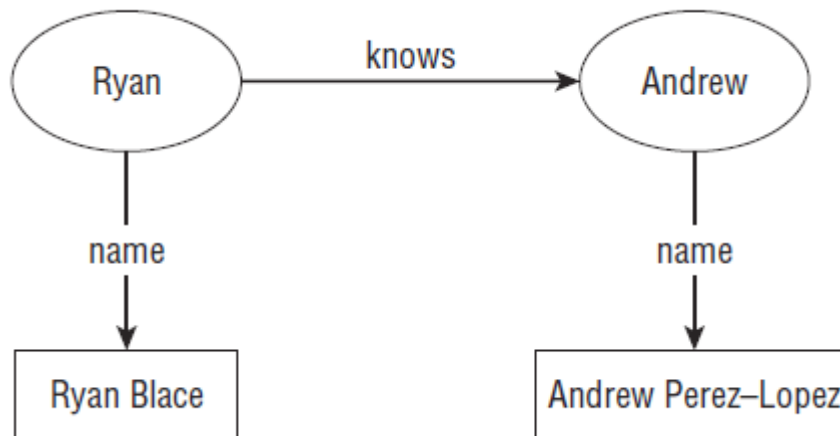
OWL for  
classification  
and reasoning

# PROPERTIES

# Properties

---

- A property in OWL is a resource that is used as a predicate in statements that describe individuals.
- Object properties
  - link individuals to other individuals
- Datatype properties
  - link individuals to literal values.



# OWL Properties


---

- **Datatype Property** – relates Individuals to data (int, string, float etc)
  - Pneumonia hasRadiologyFinding xsd:String
- **Object Property** – relates Individuals
  - BacterialPneumonia hasCause Bacterium
- **Annotation Property** – for attaching metadata to classes, individuals or properties
  - OntologyClass hasAuthor Natasha

# Datatype Properties

---

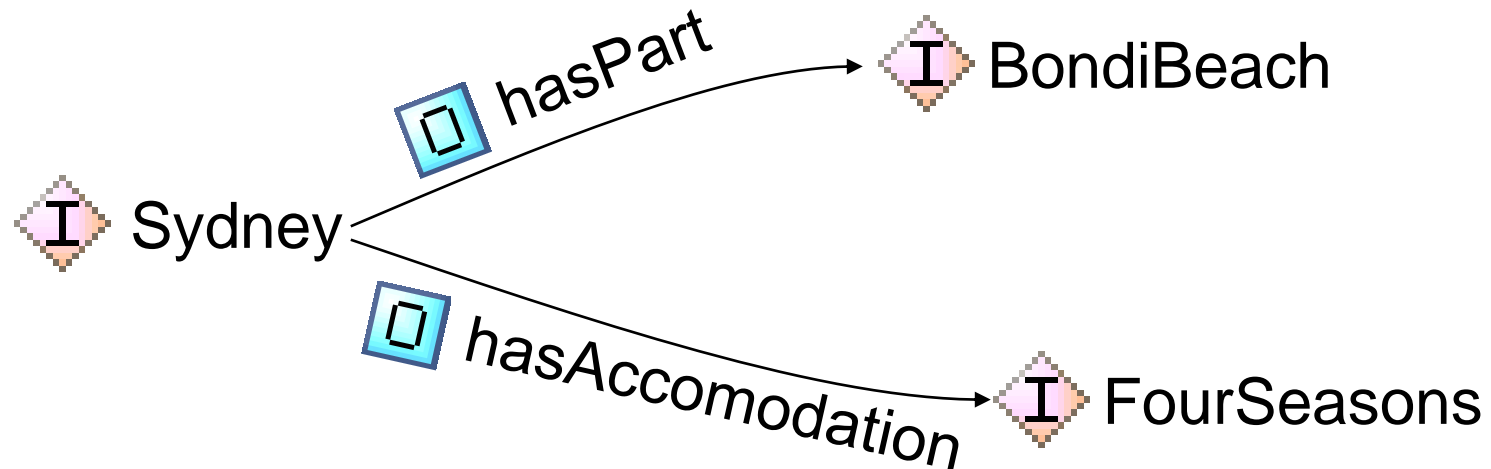
- Link individuals to primitive values  
(integers, floats, strings, booleans etc)
- Often: AnnotationProperties without formal  
“meaning”

 Sydney
hasSize = 4,500,000 isCapital = true rdfs:comment = “Don’t miss the opera house”

# Object Properties

---

- Link two individuals together
- Relationships (0..n, n..m)



# Object Property

---

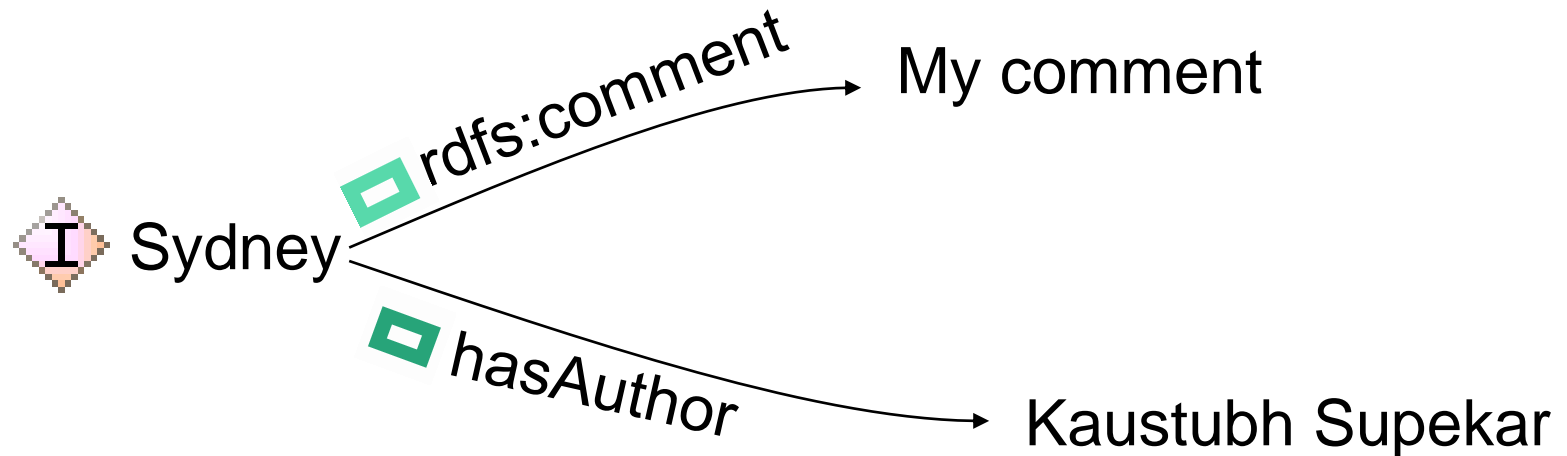
```
<owl:ObjectProperty rdf:ID="hasPart"> <rdfs:domain  
  rdf:resource="#Destination" /> <rdfs:range  
  rdf:resource="#Beaches" />  
</owl:ObjectProperty>
```



# Annotation Properties

---

- To annotate classes, properties, and individuals
- Usually used for documentation



# Annotation Properties

---

- *Annotations* are statements (triples) that have annotation properties as predicates.
- They are similar to normal OWL properties, but they have no associated semantics

PROPERTY	DESCRIPTION OF USE
<code>rdfs:label</code>	A label, or terse description of the subject resource.
<code>rdfs:comment</code>	A comment about the subject resource.
<code>owl:versionInfo</code>	Information about the subject ontology or resource version. Frequently used to embed source control metadata.
<code>rdfs:seeAlso</code>	Used to specify that another resource may hold more information about the subject resource. Not commonly used.
<code>rdfs:isDefinedBy</code>	Used to specify that another resource defines the subject resource. Not commonly used.

# Property Domain & Range

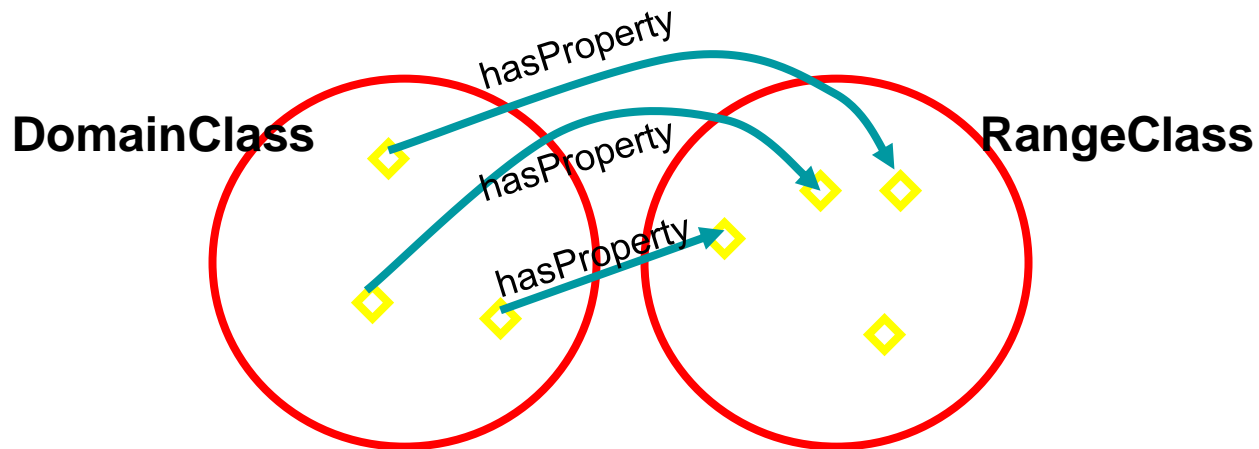
---

- allows you to describe the domain and range relationships between properties and classes or datatypes
  - **rdfs:domain**—Specifies the type of all individuals who are the subject of statements using the property being described
  - **rdfs:range**—Specifies the type of all individuals or the datatype of all literals that are the object of statements using the property being described
- Domain and range specify the class memberships of individuals and the datatypes of literals

# Property Domain & Range

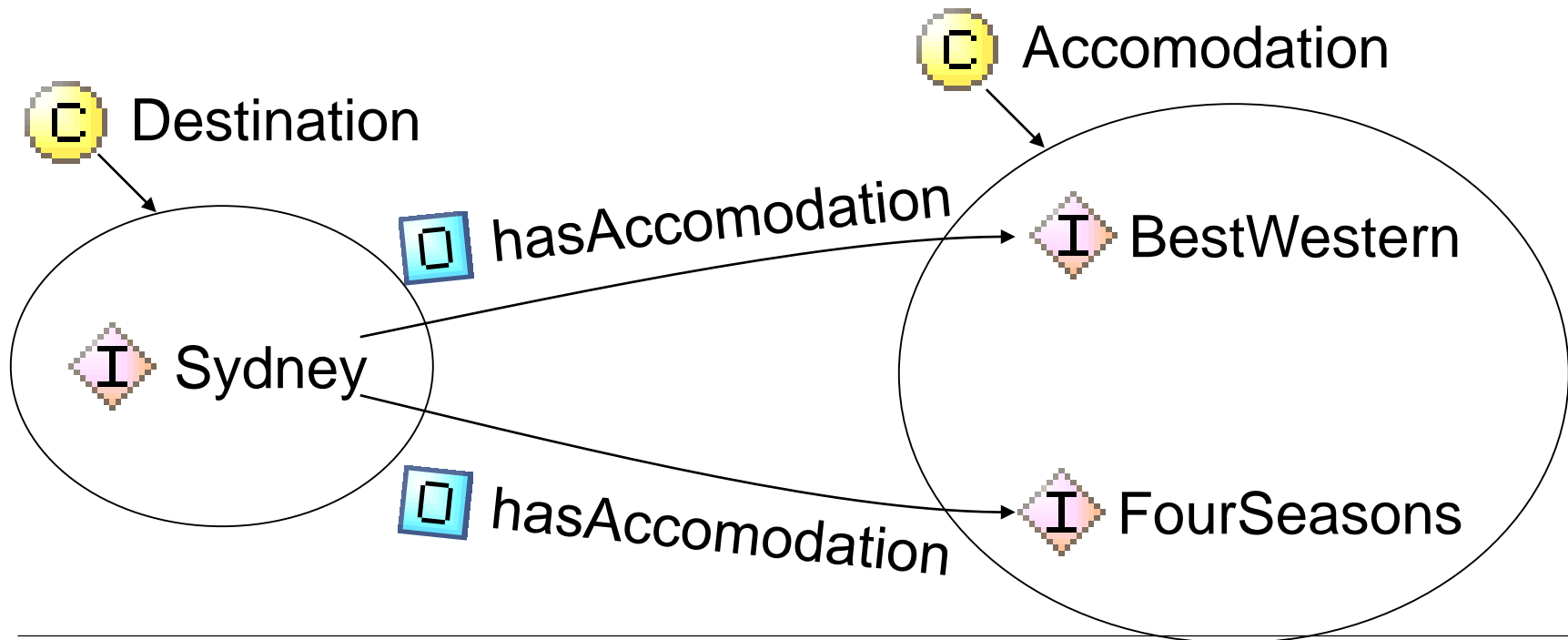
---

- If a relation is:  
subject\_individual  $\rightarrow$  hasProperty  $\rightarrow$  object\_individual
- The **domain** is the class of the **subject** individual
- The **range** is the class of the **object** individual (or a datatype if hasProperty is a Datatype Property)



# Properties, Range and Domain

- Property characteristics
  - Domain: “left side of relation” (Destination)
  - Range: “right side” (Accommodation)



# Domains

---

- Individuals can only take values of properties that have **matching domain**
  - “Only Destinations can have Accommodations”
- Domain can contain **multiple classes**
- Domain can be **undefined**: Property can be used everywhere

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

```
<owl:ObjectProperty  
rdf:ID="madeFromGrape">  
  <rdfs:range rdf:resource="#WineGrape"/>  
</owl:ObjectProperty>
```

## Sub Properties

---

- properties can be arranged into taxonomies

Property2: (Property1) rdfs:subPropertyOf (Property2)

- Property1 is a specialization of Property2.
- Any two resources related using Property1 are implicitly related by Property2.

## Sub-properties Example

---

```
<owl:ObjectProperty rdf:ID="hasWineDescriptor">  
  <rdfs:domain rdf:resource="#Wine" />  
  <rdfs:range rdf:resource="#WineDescriptor" />  
</owl:ObjectProperty>  
<owl:ObjectProperty rdf:ID="hasColor">
```

```
  <rdfs:subPropertyOf  
    rdf:resource="#hasWineDescriptor" />  
  <rdfs:range rdf:resource="#WineColor" /> ...  
</owl:ObjectProperty>
```



# Inverse Properties

---

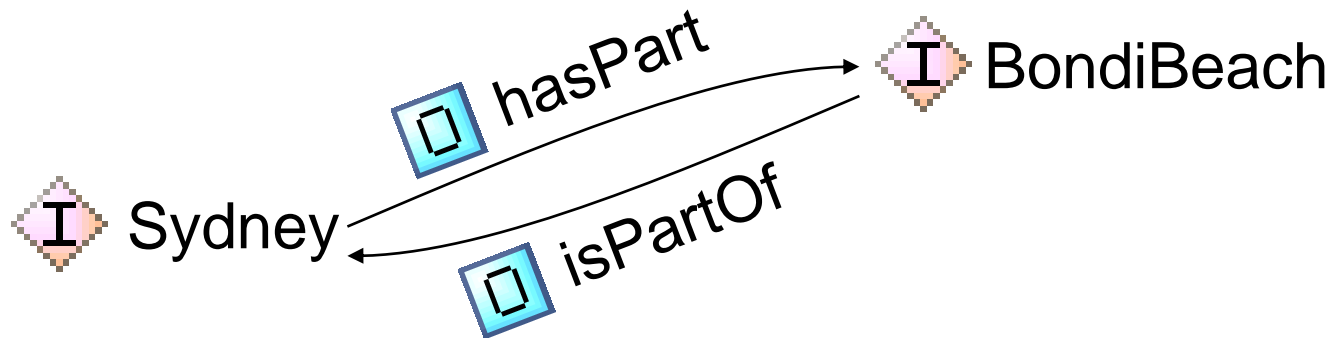
- Properties assert directed relationships, from domain to range or subject to object.
  - identifies—is identified by
  - has child—has parent
  - has part—is a part of

```
<owl:ObjectProperty rdf:ID="hasChild">  
  <owl:inverseOf rdf:resource="#hasParent"/>  
</owl:ObjectProperty>
```

# Inverse Properties

---

- Represent bidirectional relationships
- *Adding a value to one property also adds a value to the inverse property (!)*



# Inverse Properties

---

@prefix ex: <http://example.org/>.

...

ex:hasOwner rdf:type owl:ObjectProperty.

ex:owns rdf:type owl:ObjectProperty.

#has owner is the inverse of owns

ex:hasOwner owl:inverseOf ex:owns.

ex:Daisy ex:hasOwner ex:Ryan

- Both the domain and range of the owl:inverseOf relationship must be object properties !

# Disjoint Properties

---

- OWL 2
- When two properties, property1 and property2, are disjoint
  - no two statements can exist where the subjects and objects of each statement are the same

@prefix ex: <http://example.org/>.

...

ex:hasMother rdf:type owl:ObjectProperty.

ex:hasFather rdf:type owl:ObjectProperty.

ex:hasMother owl:propertyDisjointWith ex:hasFather.

# Disjoint Properties

---

- Use a construct to identify that sets of properties are pair-wise disjoint.

```
@prefix ex: <http://example.org/>.
```

```
...
```

```
ex:hasMother rdf:type owl:ObjectProperty.
```

```
ex:hasFather rdf:type owl:ObjectProperty.
```

```
[] rdf:type owl:AllDisjointProperties;
```

```
  owl:members (
```

```
    ex:hasMother
```

```
    ex:hasFather
```

```
  ).
```

# Mathematical properties of an OWL 'property'

---

- Functional
  - Person has\_Mother Mother
- InverseFunctional
  - Person has\_SSN SSN
- Symmetric
  - A worksWith B  $\implies$  B worksWith A
- Transitive
  - A hasPart B, B hasPart C  $\implies$  A hasPart C

# Functional Property

---

- A functional property is a property that can have only one (unique) value  $y$  for each instance  $x$ ,
- there cannot be two distinct values  $y_1$  and  $y_2$  such that the pairs  $(x, y_1)$  and  $(x, y_2)$  are both instances of this property.
- it can associate only a single unique value with a particular individual

# Functional Property

---

- Date of birth : an individual can have only one birthday, but two people can share the same birthday
- has biological mother: a person can have only one mother, but siblings can share the same mother
- The semantics: if a person has two functional properties, each with different individuals as the objects of the statements, it implies that the two objects are the same.



# Functional Property

---

```
<owl:ObjectProperty rdf:ID="husband">  
  <rdf:type    rdf:resource="&owl;FunctionalProperty" />  
  <rdfs:domain rdf:resource="#Woman" />  
  <rdfs:range  rdf:resource="#Man" />  
</owl:ObjectProperty>
```

-----

```
<owl:ObjectProperty rdf:ID="husband">  
  <rdfs:domain rdf:resource="#Woman" />  
  <rdfs:range  rdf:resource="#Man" />  
</owl:ObjectProperty>
```

```
<owl:FunctionalProperty rdf:about="#husband" />
```

# Inverse Functional Property

---

- owl:InverseFunctionalProperty.
- the object uniquely identifies the subject
- No two individuals can have the same value for that property
- however, an individual can have more than one unique value
- Example:
  - has email address
  - No two people can have the same email address.
- Note: In contrast to functional properties, it is completely valid for one person to have multiple email addresses

## Semantics:

- If the property has email address is inverse-functional and two people share the same value for the property, it either implies
  - an error or
  - indicates that the two are actually the same person.

# Symmetric / Asymmetric Properties

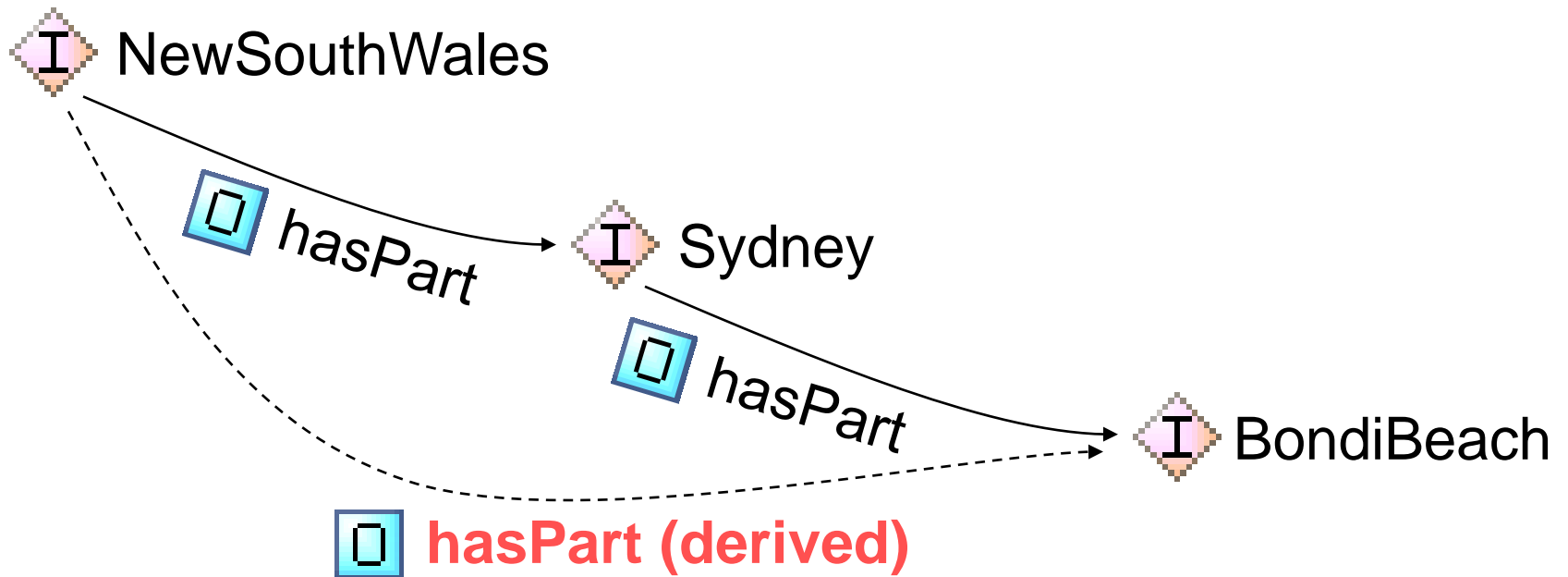
---

- owl:SymmetricProperty
- its own inverse.
- bidirectional relationships.
  - The existence of the relationship in one direction (subject to object) implies that the same relationship exists in the opposite direction (object to subject) as well.
- Example : equals, adjacent to, and has spouse.
- owl:AsymmetricProperty
- never exist as a bidirectional relationship.
- No two individuals A and B can be related (A p B) and (B p A) by an asymmetric property p.
- Example :has mother , is greater than.

# Transitive Properties

---

- If A is related to B and B is related to C then A is also related to C
- Often used for part-of relationships



# Transitive Property Example

---

```
<owl:ObjectProperty rdf:ID="locatedIn">  
  <rdf:type rdf:resource="&owl;TransitiveProperty" />  
  <rdfs:domain rdf:resource="&owl;Thing" />  
  <rdfs:range rdf:resource="#Region" />  
</owl:ObjectProperty>
```

```
<Region rdf:ID="SantaCruzMountainsRegion">  
  <locatedIn rdf:resource="#CaliforniaRegion" />  
</Region>
```

```
<Region rdf:ID="CaliforniaRegion">  
  <locatedIn rdf:resource="#USRegion" />  
</Region>
```

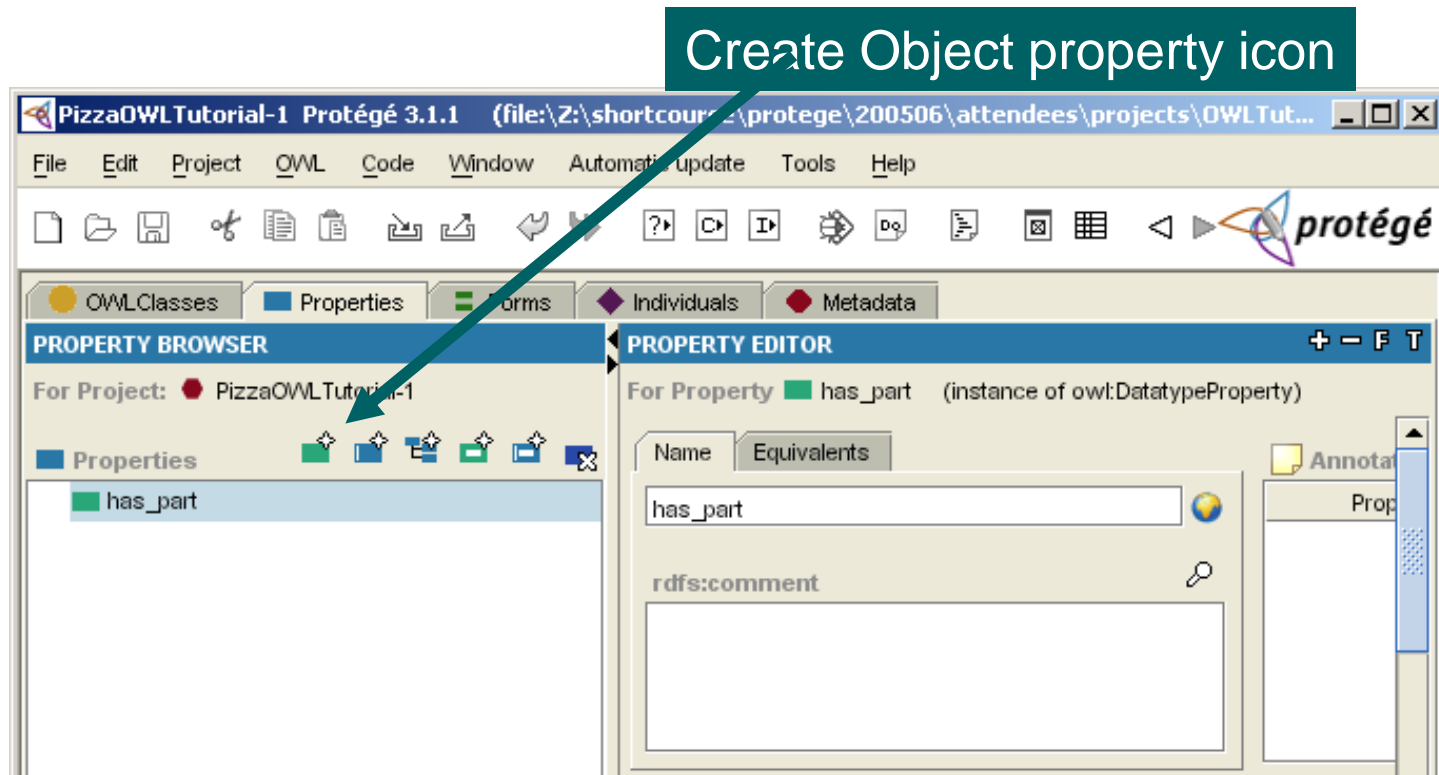
# Define Properties of Classes

---

- Properties in a class definition describe attributes of instances of the class and relations to other instances
  - Each Pneumonia will have radiology findings and a cause
  - Each cause for pneumonia will have a causative organism.

# Create object property “has\_part”

- Click on properties tab
- Click on Create\_Object\_property icon and create has\_part



# Object property hasLocus (already present)

Tutorial Protégé 3.2 beta (file: D:\DLs\STC%20OWL%20Tutorial\Tutorial.pprj, OWL / RDF Files)

File Edit Project Window Tools Help

Metadata OWLClasses Properties Individuals Forms

**PROPERTY BROWSER**

For Project: Tutorial

Object Datatype Annotation All

Object properties

- asScaleRealm
- DomainSlot
  - RelationProperty
    - hasCause ↔ causes
    - hasLocus**
    - causes ↔ hasCause
  - RefiningProperty
  - hasActionRole

**PROPERTY EDITOR**

For Property: hasLocus (instance of owl:ObjectProperty)

hasLocus (instance of owl:ObjectProperty)

**PROPERTY EDITOR**

For Property: hasLocus (instance of owl:ObjectProperty)

Property	Value	Lang
rdfs:comment	"Links disorders with OrganicStructures and OrganicProcesses"	
rdfs:label	hasLocation	

Domain: owl:Thing

Range:

☐ Functional  
☐ InverseFunctional  
☐ Symmetric  
☐ Transitive

Inverse

Annotations

Lang

Super Properties

- RelationProperty



# Datatype Property “hasRadiologyFinding”

The screenshot displays the OWL Property Editor interface. On the left, the 'PROPERTY BROWSER' shows the 'RadiologyFinding' property under 'Datatype Properties'. The main 'PROPERTY EDITOR' window is titled 'For Property: RadiologyFinding (instance of owl:D...)' and shows the 'rdfs:comment' property. Below this, the 'Domain' is set to 'owl:Thing' and the 'Range' is set to 'string'. A green callout box with the text 'Datatype = string' has an arrow pointing to the 'Range' field.

**PROPERTY BROWSER**  
For Project: Tutorial  
Object Datatype Annotation All  
Datatype Properties  
RadiologyFinding

**PROPERTY EDITOR**  
For Property: RadiologyFinding (instance of owl:D...)  
Property  
rdfs:comment  
Domain owl:Thing  
Range string  
Allowed value

Datatype = string

# Create annotation property “hasAuthor”

The screenshot displays the Protege ontology editor interface. The main window is titled "hasAuthor (instance of owl:DatatypeProperty)". Below the title bar is the "PROPERTY EDITOR" tab, which contains a table for defining the property's characteristics.

**PROPERTY EDITOR**

For Property:

Property	Value
<input checked="" type="checkbox"/> rdfs:comment	

Domain:  Range:  ☐ Functional

Allowed values:

Metadata | **OWLClasses** | Properties | Individuals | Forms

**SUBCLASS EXPLORER**

For Project: Tutorial

Asserted Hierarchy

- owl:Thing
  - DomainConcept
    - RefiningConcept
    - SelfStandingConcept
      - ActionRole
      - Physical
        - PhysicalProcess
          - OrganicProcess

**CLASS EDITOR**

For Class:  (instance of owl:Class)

Property	Value
<input checked="" type="checkbox"/> hasAuthor	Daniel Rubin
<input checked="" type="checkbox"/> rdfs:comment	This represents pneumonia caused by bacteria

Tel +49/241/8021501 | F

# INDIVIDUALS

# Individuals

---

- Represent specific things in the domain
- Two names could represent the same “real-world” individual

 Sydney

 BondiBeach

 SydneysOlympicBeach

## Example of Individuals

---

<Region rdf:ID="CentralCoastRegion" />

equivalent to:

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

# Example of Individuals

---

```
<Person rdf:ID="Adam">  
  <rdfs:label>Adam</rdfs:label>  
  <rdfs:comment>Adam is a person.</rdfs:comment>  
  <age><xsd:integer rdf:value="13"/></age>  
  <shoesize><xsd:decimal  
    rdf:value="9.5"/></shoesize>  
</Person>
```

# Create OWL instances

---

- **Create an instance of a class**
  - The class becomes a **direct type** of the instance
  - Any superclass of the direct type is a **type** of the instance
  - Generally, you create instances if you have a “type-of” something

# Class versus Individual (Instance)

---

- **Levels of representation:**
  - In certain contexts a class can be considered an instance of something else.
  - Grape, set of all *grape varieties*. CabernetSauvignonGrape is an instance of this class, but could be considered a class, the set of all actual Cabernet Sauvignon grapes.
- **Subclass vs. instance:** easy to confuse *instance-of* relationship with *subclass* relationship!
  - CabernetSauvignonGrape as individual & instance of Grape, or subclass of Grape.
  - But: Grape class is the set of all *grape varieties*, any subclass should be a subset.
  - CabernetSauvignonGrape is an *instance of* Grape, It does not describe a subset of Grape varieties, it *is* a grape varietal.



# RESTRICTIONS

# Restrictions (Overview)

---

- A restriction describes a class of individuals based on the relationships that members of the class participate in.
- A restriction is a kind of class, in the same way that a named class is a kind of class
- Semantic Web: information that has been specified in one context can be reused either by others in different contexts
- Restriction Types:
  - Data Types Restriction
  - Property Restriction
    - Value Restrictions: owl:allValuesFrom (Universal Restrictions), owl:someValuesFrom (Existential Restrictions), and owl:hasValue.
    - Cardinality Restrictions: owl:minCardinality, owl:maxCardinality, owl:cardinality

# Data Types Restrictions

---

- Ranges of data values that are identified using URIs.
- OWL allows you to use a number of predefined datatypes
  - Mostly defined in the XML Schema Definition (xsd) namespace
- Numeric—xsd:integer, xsd:float, xsd:real, xsd:decimal
- String—xsd:string, xsd:token, xsd:language
- Boolean—xsd:Boolean
- URI—xsd:anyUri
- XML—rdf:XMLLiteral
- Time—xsd:dateTime
- Define new data type:
  - you can create a custom data range
  - you can define a datatype in terms of other datatypes

# Data Types Restrictions

---

- define your own datatypes by creating instances of the class `rdfs:Datatype`
- Associate one or more facet restrictions :
  - describing a set of values for a specific datatype that makes up the range of valid values

FACET	DESCRIPTION
<code>xsd:length</code>	<i>N</i> is the exact number of items (or characters) allowed.
<code>xsd:minLength</code>	<i>N</i> is the minimum number of items (or characters) allowed.
<code>xsd:maxLength</code>	<i>N</i> is the maximum number items (or characters) allowed.
<code>xsd:pattern</code>	A regular expression that defines allowed character strings.
<code>xsd:minInclusive</code>	Values must be greater than or equal to <i>N</i> .
<code>xsd:minExclusive</code>	Values must be strictly greater than <i>N</i> .
<code>xsd:maxInclusive</code>	Values must be less than or equal to <i>N</i> .
<code>xsd:maxExclusive</code>	Values must be strictly less than <i>N</i> .
<code>xsd:totalDigits</code>	The number of digits must be equal to <i>N</i> .
<code>xsd:fractionDigits</code>	<i>N</i> is the maximum number of decimal places allowed.

*N* refers to the value portion of the facet restriction.

# Data Types Restrictions

---

@prefix ex: <http://example.org/>.

...

#integers in the range (5, 10]

[]

```
  rdf:type rdfs:Datatype;  
  owl:onDatatype xsd:integer;  
  owl:withRestrictions (  
    [  
      xsd:maxInclusive 10;  
    ]  
    [  
      xsd:minExclusive 5;  
    ]  
  ).
```

#valid social security numbers

[]

```
  rdf:type rdfs:Datatype;  
  owl:onDatatype xsd:string;  
  owl:withRestrictions (  
    [  
      xsd:pattern "[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9]";  
    ]  
  ).
```

---

# Data Types Restrictions

---

Define datatypes in terms of other datatypes

- owl:intersectionOf,
- owl:unionOf
- owl:datatypeComplementOf

@prefix ex: <http://example.org/>.

...

# Integers other than 0 using union-of

```
[] rdf:type rdfs:Datatype;
```

```
owl:unionOf (
```

```
[
```

```
  rdf:type rdfs:Datatype;
```

```
  owl:onDatatype xsd:integer;
```

```
  owl:withRestrictions (
```

```
    [
```

```
      xsd:maxExclusive 0;
```

```
    ]
```

```
  )
```

```
]
```

```
[
```

```
  rdf:type rdfs:Datatype;
```

```
  owl:onDatatype xsd:integer;
```

```
  owl:withRestrictions (
```

```
    [
```

```
      xsd:minExclusive 0;
```

```
    ]
```

```
  )
```

```
]
```

```
).
```







# Property Restrictions

---

- A property restriction describes the class of individuals that meet the specified property-based conditions.
- The restriction is declared using the construct `owl:Restriction`
- The property to which the restriction refers is identified using the property `owl:onProperty`
- Restrictions are applied to a particular class by stating that the class is either a subclass (`rdfs:subClassOf`) or the equivalent class (`owl:equivalentClass`) of the restriction
  - `owl:equivalentClass` is a construct that states that two classes are the same and have the same class extension
- class members must meet the conditions of the restriction *and* any individual who meets the conditions of the restriction is implicitly a member of the class.
- A single class can contain many restrictions.

# Property Restrictions

---

- An **anonymous class** consisting of all individuals that fulfill the condition
- Define a condition for property values
  - allValuesFrom 
  - someValuesFrom 
  - hasValue 
  - minCardinality 
  - maxCardinality 
  - cardinality 



# Property Restrictions: Value Restrictions







---

RESTRICTION	INTERPRETATION
<code>owl:allValuesFrom</code>	For all instances, if they have the property, it must have the specified range.
<code>owl:someValuesFrom</code>	For all instances, they must have at least one occurrence of the property with the specified range.
<code>owl:hasValue</code>	For all instances, they must have an occurrence of the property with the specified value.

This table is derived from an example contained in the OWL Web Ontology Language Guide—<http://www.w3.org/TR/owl-guide>.

## someValuesFrom Restrictions










- Meaning: At least one value of the property must be of a certain type
- Others may exist as well
- Example: A NationalPark is a RuralArea that has at least one Campground and offers at least one Hiking opportunity

Asserted Conditions	
	NECESSARY & SUFFICIENT
	NECESSARY
 RuralArea	
 hasAccommodation Campground	
 hasActivity Hiking	

# allValuesFrom Restrictions

---

- Meaning: All values of the property must be of a certain type
- Warning: Also individuals with no values fulfill this condition (trivial satisfaction)
- Example: Hiking is a Sport that is only possible in NationalParks

Asserted Conditions						
		NECESSARY & SUFFICIENT				
		NECESSARY				
	Sports					
	$\forall$ isPossibleIn NationalPark					

# Value constraints

---

<owl:Restriction>

<owl:onProperty rdf:resource="#hasParent" />

<owl:allValuesFrom rdf:resource="#Human" />

</owl:Restriction>

# Property Restrictions: Value Restrictions

---

- A selection committee for a job should have at least one female member

:SelectionCommittee

a owl:Class ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:onProperty :committeeMember ;

owl:allValuesFrom :Person

].

rdfs:subClassOf

[ a owl:Restriction ;

owl:onProperty :committeeMember ;

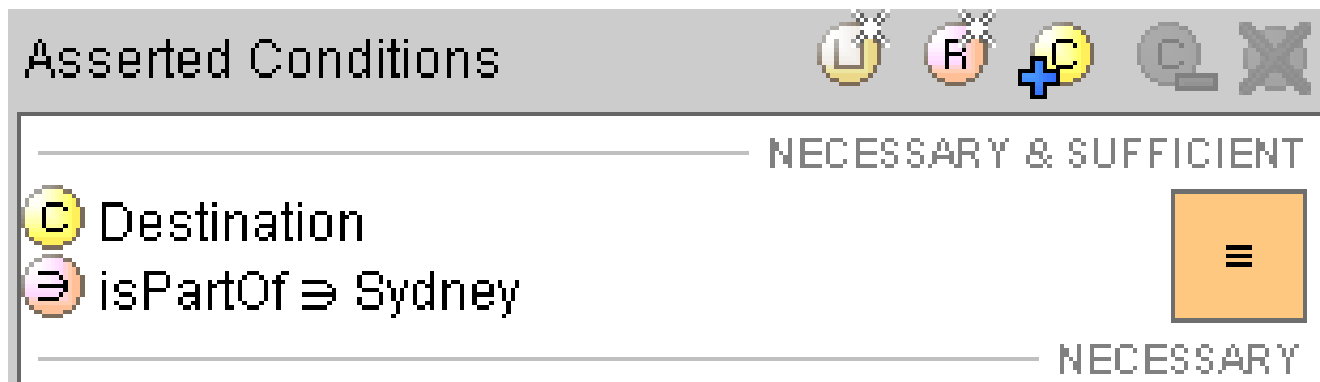
owl:someValuesFrom :FemalePerson

].

a selection committee has members who are persons and must have at least one female person as its member.

## hasValue Restrictions

- Meaning: At least one of the values of the property is a certain value
- Similar to someValuesFrom  $\exists$  but with Individuals and primitive values
- Example: A PartOfSydney is a Destination where one of the values of the isPartOf property is Sydney



# Property Restrictions: Value Restrictions

---

class definition for ex:PetsOfRyan.

@prefix ex: <http://example.org/>.

...

ex:Mammal rdf:type owl:Class.

ex:hasOwner rdf:type owl:ObjectProperty.

...

ex:PetsOfRyan rdf:type owl:Class;

    rdfs:subClassOf ex:Mammal;

    rdfs:subClassOf[

        rdf:type owl:Restriction;

        owl:onProperty ex:hasOwner;

        owl:hasValue ex:Ryan

    ].

- that all members must be mammals,
- the second is that each member must be owned by Ryan.

# Property Restrictions: Cardinality Restrictions

---




- Cardinality restrictions give you the ability to specify in very precise terms **how many times a property** can be used to describe an instance of a class.

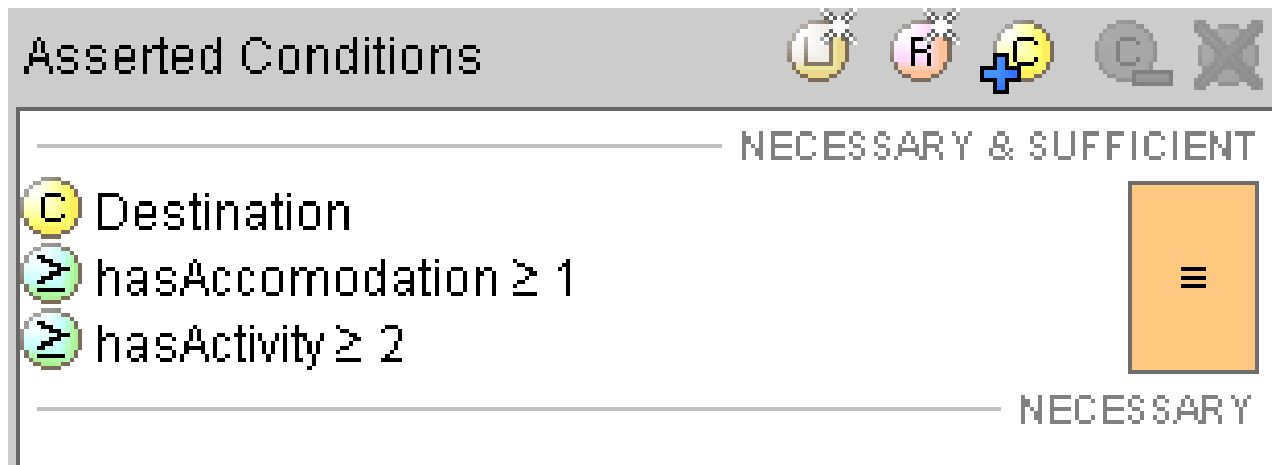
RESTRICTION	INTERPRETATION
<code>owl:minCardinality</code>	There must be at least $N$ properties.
<code>owl:maxCardinality</code>	There can be at most $N$ properties.
<code>owl:cardinality</code>	There are exactly $N$ properties.

$N$  refers to the value of the cardinality restriction. ( $N$  must be nonnegative.)



# Property Restrictions: Cardinality Restrictions

- Meaning: The property must have at least/at most/exactly x values
-  is the shortcut for  and 
- Example: A FamilyDestination is a Destination that has at least one Accomodation and at least 2 Activities



# Property Restrictions: Cardinality Restrictions

---

- Cardinality restrictions give you the ability to specify in very precise terms **how many times a property** can be used to describe an instance of a class.

```
<rdfs:subClassOf>
  <owl:Restriction owl:cardinality="1">
    <owl:onProperty rdf:resource="#hasBiologicalFather"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#shoesize"/>
    <owl:minCardinality>1</owl:minCardinality>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Any person must have exactly 1 biological father and at least one shoe size.

---

# Property Restrictions: Cardinality Restrictions

```
<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>
```

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#madeFromGrape"/>
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction>
```

# Property Restrictions: Cardinality Restrictions

---

- Cardinality and value restrictions can be combined
- Specific the class: a value restriction with a custom datatype + a cardinality restriction  
@prefix ex: <http://example.org/>.

...

# Large breeds must have average weight greater than or equal to 50 lbs

```
ex:LargeBreed rdf:type owl:Class;  
              rdfs:subClassOf ex:Breed;  
              rdfs:subClassOf [  
                rdf:type owl:Restriction;  
                owl:minCardinality 1;  
                owl:onProperty ex:averageWeight  
              ];  
              rdfs:subClassOf [  
                rdf:type owl:Restriction;  
                owl:onProperty ex:averageWeight;  
                owl:allValuesFrom [  
                  rdf:type rdfs:Datatype;  
                  owl:onDatatype xsd:real;  
                  owl:withRestrictions (  
                    [ xsd:minInclusive 50.0; ]) ]  
              ]  
].
```

---

# Define Constraints : OWL Restrictions

---

- **Quantifier** restriction
  - How to represent the fact that every pneumonia must be located in a lung?
- **Cardinality** restrictions
  - How to represent that a lung must have 3 lobes as parts ?
- **hasValue** restrictions
  - How to define the value of a relation for a class ?  
(relationship between class and a individual)

# Creating Restrictions

Restricted Property

Restriction  
Type

Filler  
Expression

Expression  
Construct  
Palette

Syntax  
check

Restricted Property

- hasBase
- hasCountryOfOrigin
- hasGreasyness
- hasIngredient
- hasSpiciness
- hasTopping
- isBaseOf
- isIngredientOf

OWLRestriction

- allValuesFrom
- someValuesFrom
- hasValue
- cardinality
- minCardinality
- maxCardinality

Filler

PizzaBase

Expression Construct Palette

OK Cancel

# Create a restriction: using a datatype property

The image shows two overlapping windows from the Protégé 3.2 beta application. The 'Create Restriction' dialog is in the foreground, and the 'CLASS EDITOR' for the 'Pneumonia' class is in the background.

**Create Restriction Dialog:**

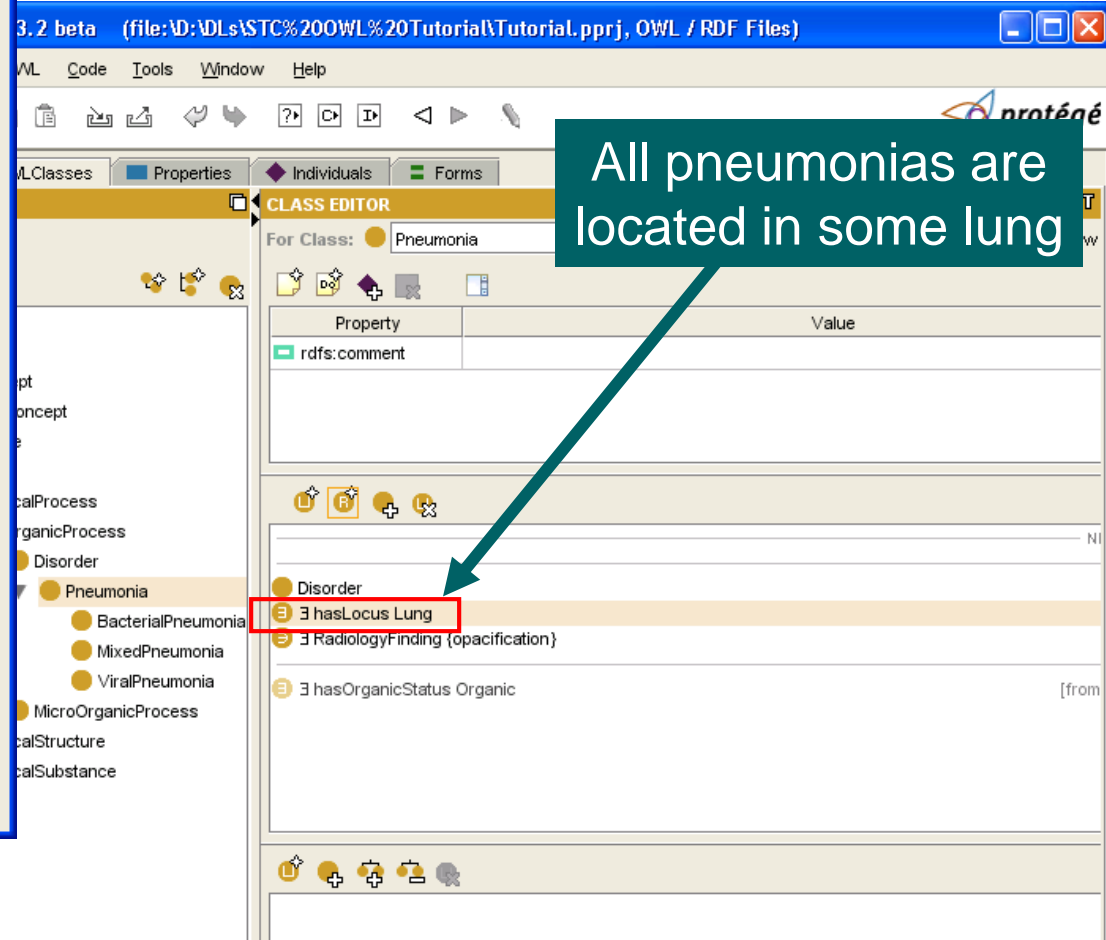
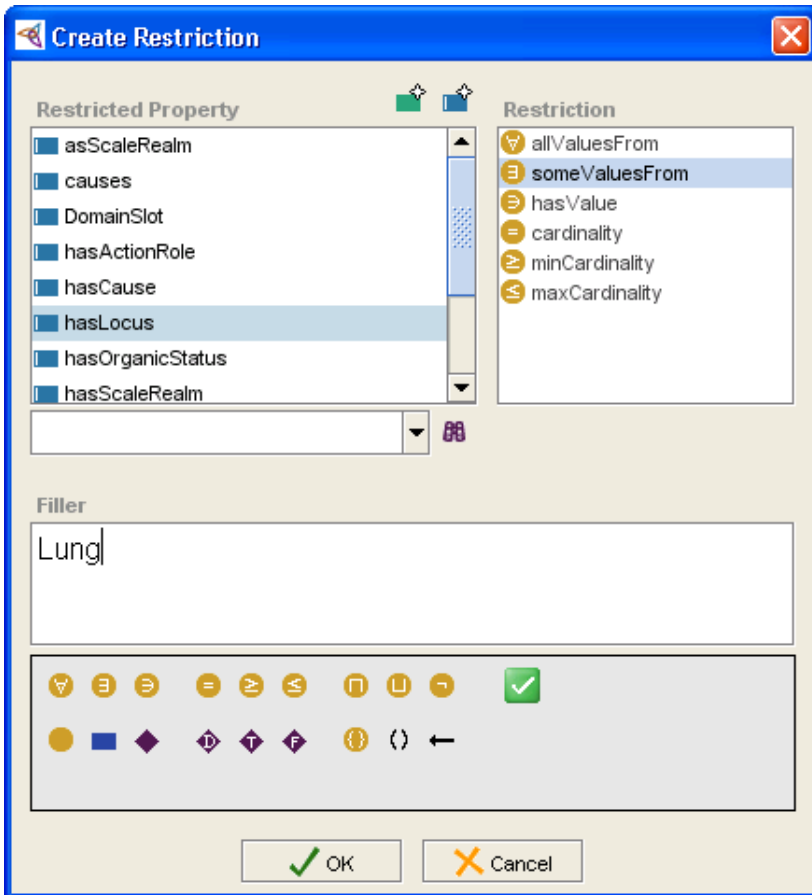
- Restricted Property:** A list of properties including `hasCause`, `hasLocus`, `hasOrganicStatus`, `hasScaleRealm`, `ModifierProperty`, `RadiologyFinding` (highlighted), `RefiningProperty`, and `RelationProperty`.
- Restriction:** A list of restriction types including `allValuesFrom`, `someValuesFrom`, `hasValue` (highlighted), `cardinality`, `minCardinality`, and `maxCardinality`.
- Filler:** A text field containing the word `opacification`.
- Buttons:** `OK` and `Cancel` buttons at the bottom.

**CLASS EDITOR (Pneumonia):**

- For Class:** `Pneumonia` (instance of owl:Class).
- Property Table:** A table with two columns: `Property` and `Value`. It contains one row: `rdfs:comment`.
- Restrictions:** A list of restrictions for the class, including `3 RadiologyFinding {Opacification}` and `3 hasOrganicStatus Organic`.

***“All pneumonias are disorders that have a radiological finding of opacification”***

# Create a restriction: using an object property



***“All pneumonias are disorders that are located in some lung and have a radiological finding of opacification”***



## ... more object properties

---

- BacterialPneumonia is caused by some bacteria
  - BacterialPneumonia  $\sqsubseteq$  causedBy some Bacteria
  - BacterialPneumonia  $\rightarrow \exists$  causedBy.Bacteria
- ViralPneumonia is caused by some virus
  - ViralPneumonia  $\sqsubseteq$  causedBy some Virus
- MixedPneumonia is caused by some bacteria and by some virus
  - MixedPneumonia  $\sqsubseteq$  (causedBy some Bacteria)  $\sqcap$  (causedBy some Virus)

# Advanced Class Description

# Class Descriptions

---

- OWL distinguishes six types of class descriptions: a class identifier (a URI reference)
  - an exhaustive enumeration of individuals that together form the instances of a class
  - a property restriction
  - the intersection of two or more class descriptions
  - the union of two or more class descriptions
  - the complement of a class description
- The first type is special in the sense that it describes a class through a *class name*.
- The other five types of class descriptions describe an *anonymous class* by *placing constraints on the class extension*.

# Named Classes

---

- A type 1 class description is syntactically represented as an named instance of owl:Class, a subclass of rdfs:Class:
  - `<owl:Classrdfs:ID="Human"/>`
- The other five forms of class descriptions consist of anonymous or unnamed classes
  - a set of RDF triples in which a blank node represents the class being described
  - That blank node has an rdf:typeproperty whose value is owl:Class.

# Enumerated Classes






- Consist of exactly the listed individuals
- No individual that is not listed in the enumeration can become a member of this class.

 OneStarRating


 TwoStarRating


 ThreeStarRating


Asserted Conditions

NECESSARY & SUFFICIENT

 Accomodation

  $\exists$  hasRating {OneStarRating TwoStarRating}



NECESSARY

 BudgetAccomodation

## Example Description: Enumeration

---

<owl:Class>

<owl:oneOf rdf:parseType="Collection">

<owl:Thing rdf:about="#Eurasia"/>

<owl:Thing rdf:about="#Africa"/>

<owl:Thing rdf:about="#NorthAmerica"/>

<owl:Thing rdf:about="#SouthAmerica"/>

<owl:Thing rdf:about="#Australia"/>

<owl:Thing rdf:about="#Antarctica"/> </owl:oneOf>

</owl:Class>

# OWL Class Constructors

Constructor	DL Syntax	Example	Modal Syntax
intersectionOf	$C_1 \sqcap \dots \sqcap C_n$	Human $\sqcap$ Male	$C_1 \wedge \dots \wedge C_n$
unionOf	$C_1 \sqcup \dots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1 \vee \dots \vee C_n$
complementOf	$\neg C$	$\neg$ Male	$\neg C$
oneOf	$\{x_1\} \sqcup \dots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x_1 \vee \dots \vee x_n$
allValuesFrom	$\forall P.C$	$\forall$ hasChild.Doctor	$[P]C$
someValuesFrom	$\exists P.C$	$\exists$ hasChild.Lawyer	$\langle P \rangle C$
maxCardinality	$\leq_n P$	$\leq 1$ hasChild	$[P]_{n+1}$
minCardinality	$\geq_n P$	$\geq 2$ hasChild	$\langle P \rangle_n$

- XMLS **datatypes** as well as classes in
- Arbitrarily complex **nesting** of constructors

# OWL Axioms

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$ Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq \neg\{peter\}$
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent <sup>-</sup>
transitiveProperty	$P^+ \sqsubseteq P$	ancestor <sup>+</sup> $\sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leq 1P$	$\top \sqsubseteq \leq 1$ hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leq 1P^-$	$\top \sqsubseteq \leq 1$ hasSSN <sup>-</sup>

Axioms are used to associate class and property IDs with either partial or complete specifications of their characteristics, and to give other logical information about classes and properties.



## Set Operators

---

- describe the membership of a class in terms of the extensions of other classes:

SET OPERATION	INTERPRETATION
<code>owl:intersectionOf</code>	Individuals that are instances of all classes A, B, and C
<code>owl:unionOf</code>	Individuals that are instances of at least one class A, B, or C
<code>owl:complementOf</code>	Individuals that are not instances of class A

A, B, and C are class expressions (class reference, class definition, restriction, etc.).

# Logical Class Definitions

---

- Define classes out of other classes



– unionOf (or)



– intersectionOf (and)



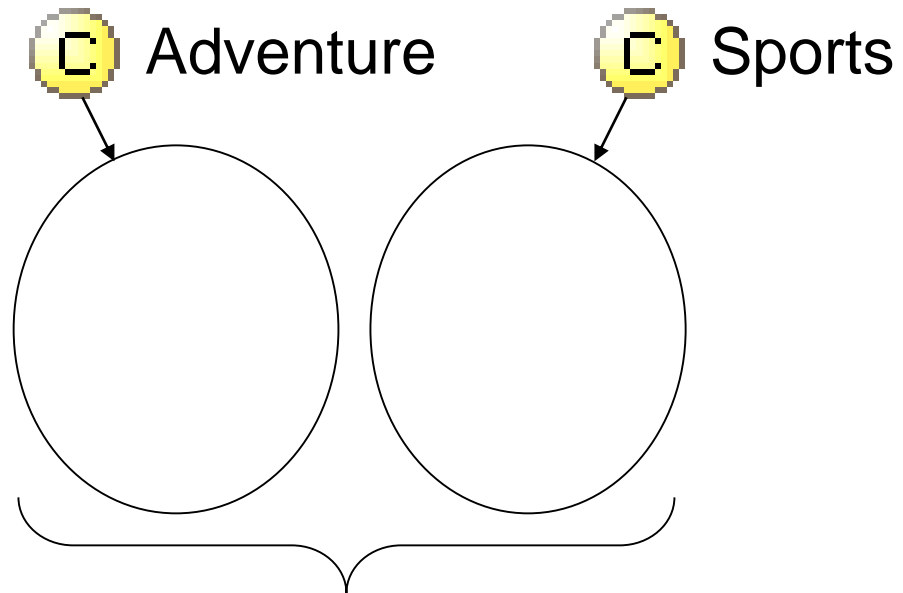
– complementOf (not)

- Allow arbitrary nesting of class descriptions (A and (B or C) and not D)
- >>> OWL DL

## unionOf

---

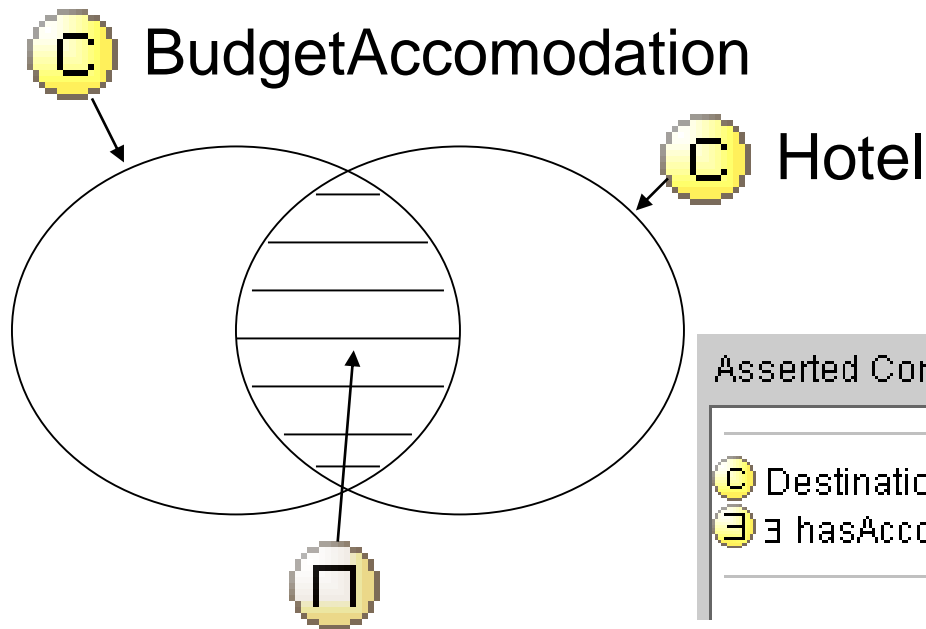
- The class of individuals that belong to class A **or** class B (or both)
- Example: Adventure or Sports activities



  $\exists$  hasActivity (Sports  $\sqcup$  Adventure)

# intersectionOf

- The class of individuals that belong to both class A **and** class B
- Example: A BudgetHotelDestination is a destination with accomodation that is a budget accomodation **and** a hotel



Asserted Conditions

NECESSARY & SUFFICIENT

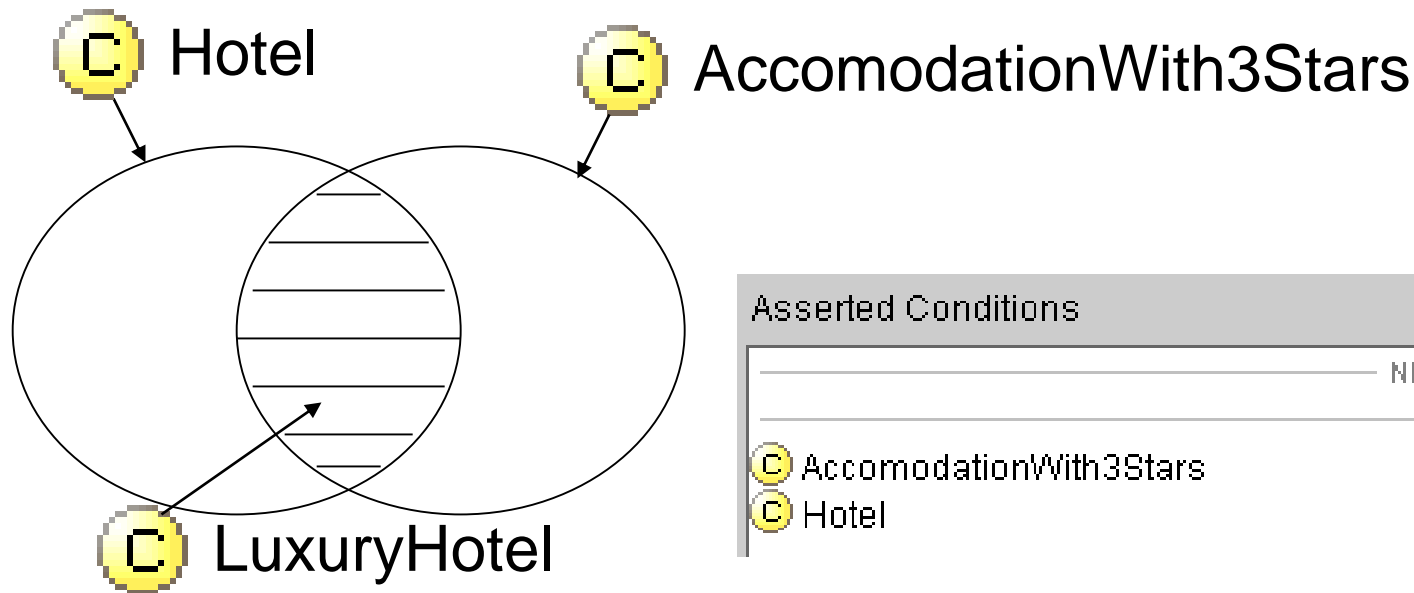
Destination










$\exists$  hasAccommodation (BudgetAccommodation  $\cap$  Hotel)

NECESSARY

# Implicit intersectionOf

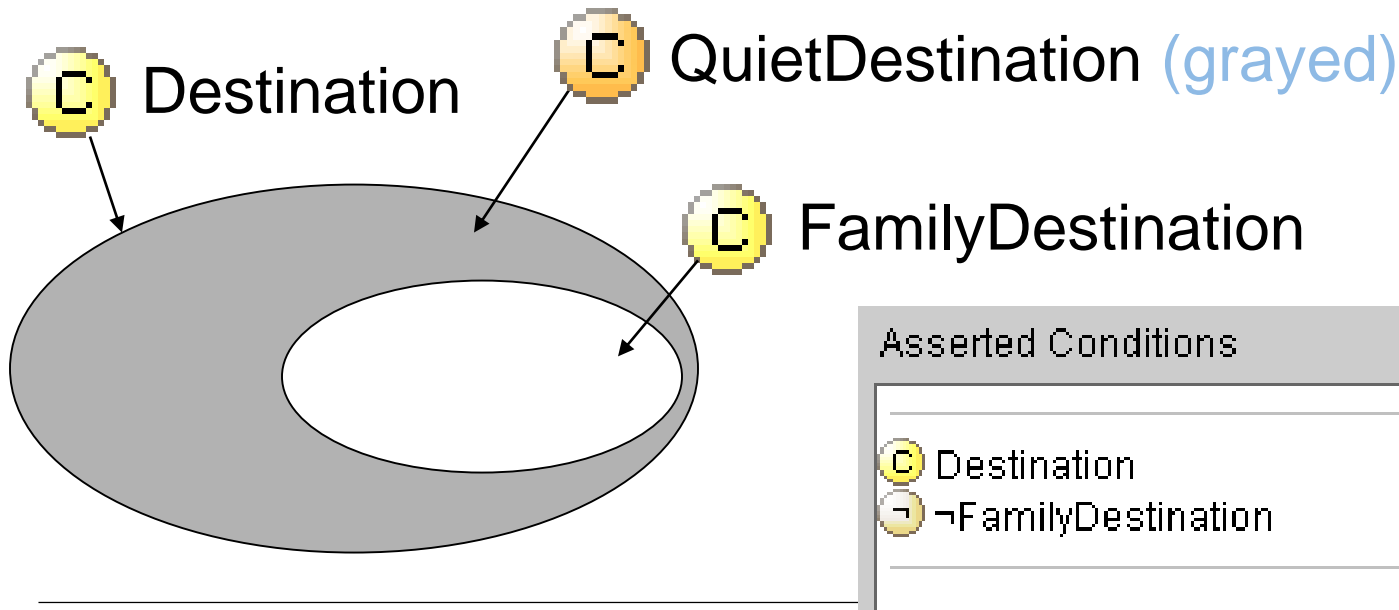
- When a class is defined by more than one class description, then it consists of the intersection of the descriptions
- Example: A luxury hotel is a hotel that is also an accommodation with 3 stars



Asserted Conditions		    	
		NECESSARY & SUFFICIENT	
		NECESSARY	
	AccommodationWith3Stars		
	Hotel		

# complementOf

- The class of all individuals that do not belong to a certain class
- Example: A quiet destination is a destination that is **not** a family destination



Asserted Conditions

NECESSARY & SUFFICIENT

Destination

$\neg$ FamilyDestination

NECESSARY

# Class Conditions

---



## **Necessary Conditions:**

(Primitive / partial classes)

“If we know that something is a X,  
then it must fulfill the conditions...”



## **Necessary & Sufficient Conditions:**

(Defined / complete classes)






“If something fulfills the conditions...,  
then it is an X.”

- ClassA is a subclass of ClassB if all individuals in ClassA are also in ClassB.
- if all of the individuals in ClassB fulfil the necessary & sufficient conditions on ClassA, all of them must also be members of ClassA, and we can infer that ClassB is a subclass of ClassA

## Class Conditions (2)


 NationalPark →


Asserted Conditions





NECESSARY & SUFFICIENT


NECESSARY


 RuralArea

 ∃ hasAccommodation Campground

 ∃ hasActivity Hiking












(not everything that fulfills these conditions is a NationalPark)


 QuietDestination ↔


Asserted Conditions




NECESSARY & SUFFICIENT

NECESSARY

 Destination

 ¬FamilyDestination



(everything that fulfills these conditions is a QuietDestination)



# Classification

## C NationalPark

Asserted Conditions

NECESSARY & SUFFICIENT

NECESSARY

- C RuralArea
- E  $\exists$  hasAccommodation Campground
- E  $\exists$  hasActivity Hiking

- A RuralArea is a Destination
- A Campground is BudgetAccommodation
- Hiking is a Sport
- Therefore:  
Every NationalPark is a Backpackers-Destination

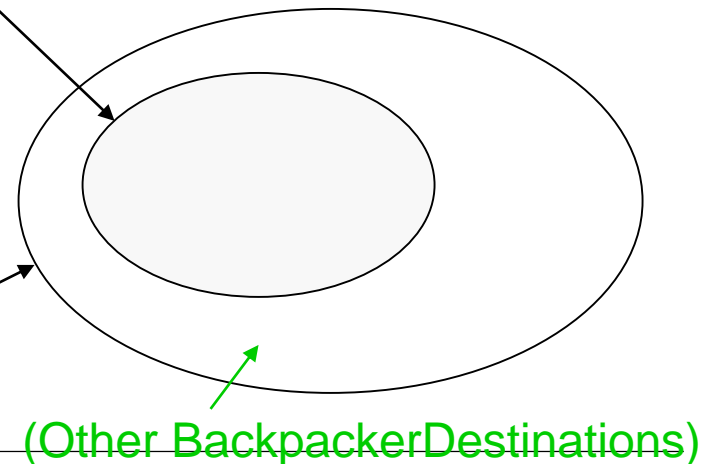
## C BackpackersDestination

Asserted Conditions

NECESSARY & SUFFICIENT

NECESSARY

- C Destination
- E  $\exists$  hasAccommodation BudgetAccommodation
- E  $\exists$  hasActivity (Sports  $\sqcup$  Adventure)



# Reasoning with Property, Domain & Range

---

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
```

```
  <rdfs:domain rdf:resource="#Wine"/>
```

```
  <rdfs:range rdf:resource="#WineGrape"/>
```

```
</owl:ObjectProperty>
```

```
<owl:Thing rdf:ID="LindemansBin65Chardonnay">
```

```
  <madeFromGrape rdf:resource="#ChardonnayGrape" />
```

```
</owl:Thing>
```

=> LindemansBin65Chardonnay is a wine

# Reasoning

# Reasoners

---

- Reasoners (“classifiers”) infer information that is not explicitly contained within the ontology
- Standard reasoner services are:
  - **Consistency Checking** (i.e., satisfiability—can a class have any instances?)
  - **Subsumption Checking** (Finding subclasses—is A a subclass of B?)
  - **Equivalence Checking**
  - **Instantiation Checking** (Which classes does an individual belong to)
- For Protégé Pellet in pre-configured (but other tools with DIG support work too)
- Reasoners can be used at runtime in applications as a querying mechanism
- Used during development as an ontology “**compiler**”. Ontologies can be compiled to check if the meaning is what was intended

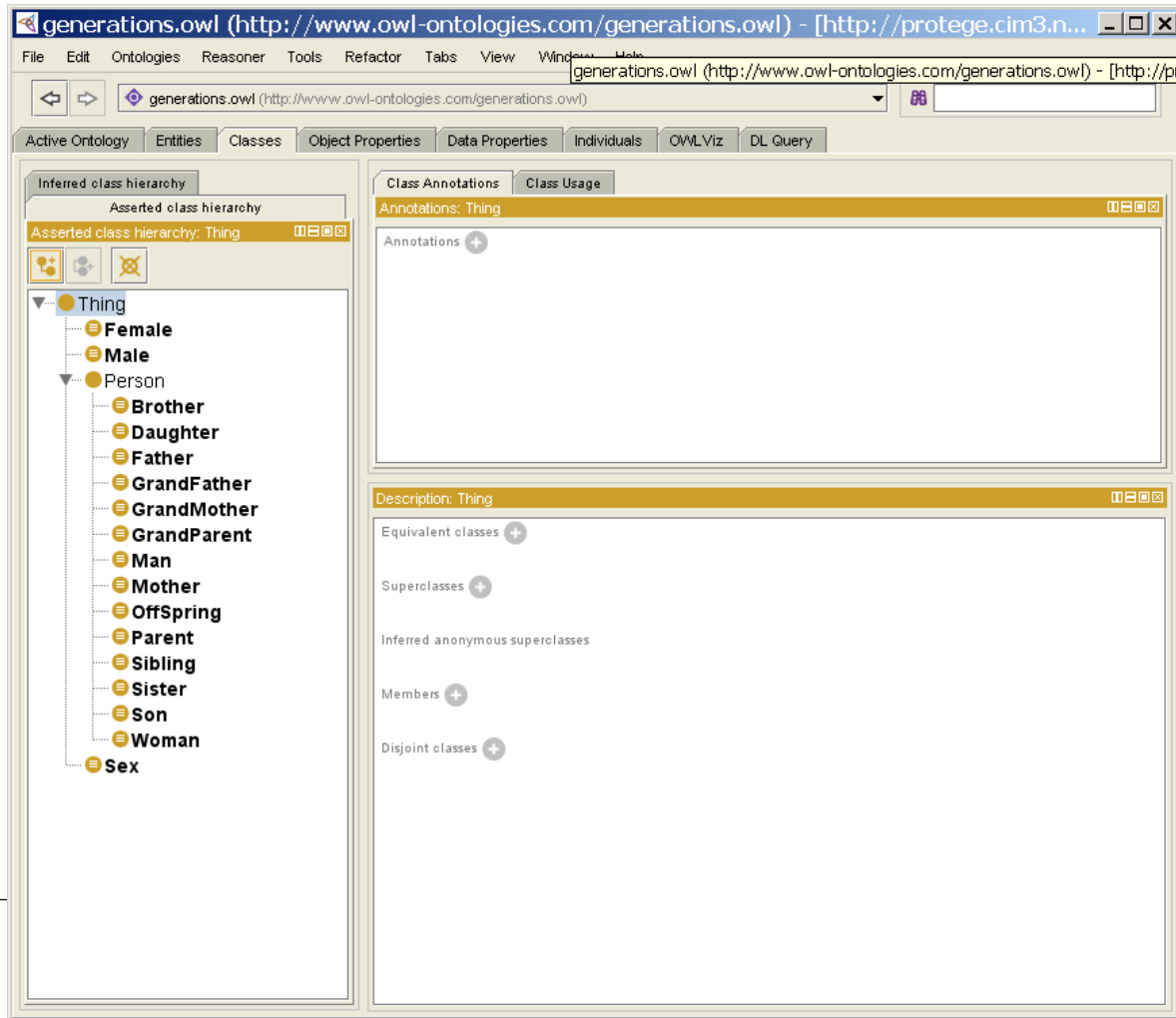
# Run a DL Reasoner with Protégé OWL

---

- Protégé OWL can work with multiple reasoners
  - Racer (<http://www.racer-systems.com/>)
  - Pellet (<http://www.mindswap.org/2003/pellet/>)
  - Fact++ (<http://owl.man.ac.uk/factplusplus/>)
- Need to install, configure, and run at least one reasoner
  - Pellet comes pre-configured
- Protégé OWL and reasoner exchange information through inter-process communication

# Playing with Protégé and Fact++

- Let's load an ontology that represent family relationships  
<http://protege.cim3.net/file/pub/ontologies/generations/generations.owl>



# Playing with Protégé and Fact++

- For example the ontology directly defines the concepts of parent and father/mother but not the relation among them

Three screenshots from Protégé showing the class descriptions for 'Parent', 'Father', and 'Mother'.

**Description: Parent**

- Equivalent classes +
- **Person**  
and hasChild some Person
- Superclasses +
- Inferred anonymous superclasses
- Members +
- Disjoint classes +

**Description: Father**

- Equivalent classes +
- **Person**  
and hasChild some Person  
and hasSex value MaleSex
- Superclasses +
- Inferred anonymous superclasses
- Members +
- Disjoint classes +

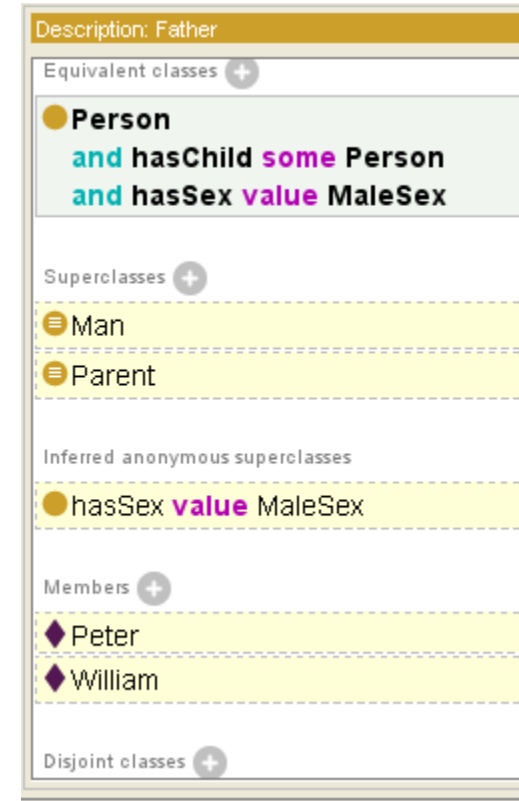
**Description: Mother**

- Equivalent classes +
- **Person**  
and hasChild some Person  
and hasSex value FemaleSex
- Superclasses +
- Inferred anonymous superclasses
- Members +
- Disjoint classes +

- What happens if we attach the Fact++ Reasoner?

# Playing with Protégé and Fact++

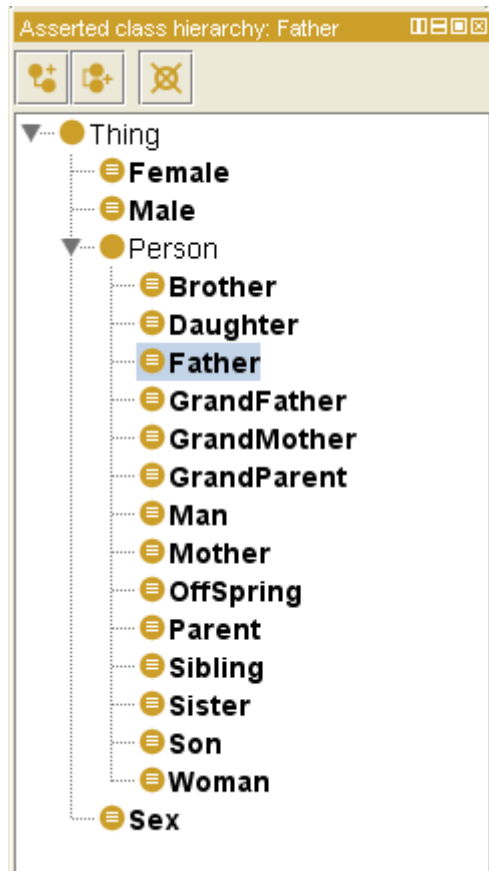
- Father now is classified as subclass of Man and Parent
- Two instances are part of this class
  - William type Person
    - hasChild Peter
    - hasSex Male
  - Peter type Person
    - hasChild Matt
    - hasSex Male



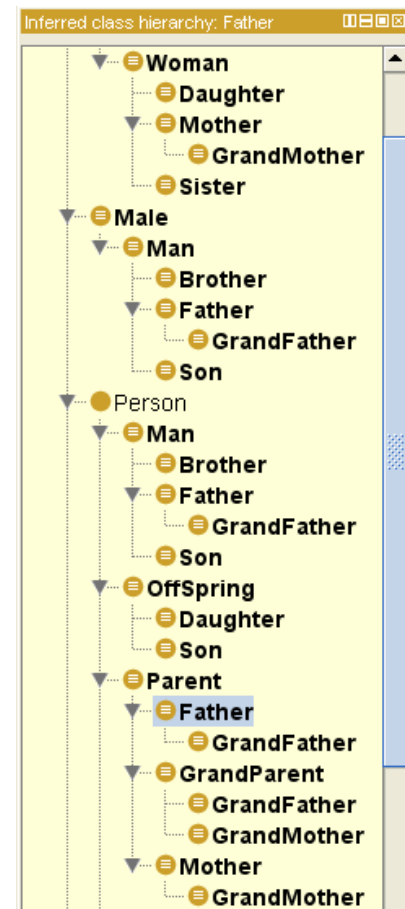


# Playing with Protégé and Fact++

- Asserted Hierarchy



- Inferred Hierarchy



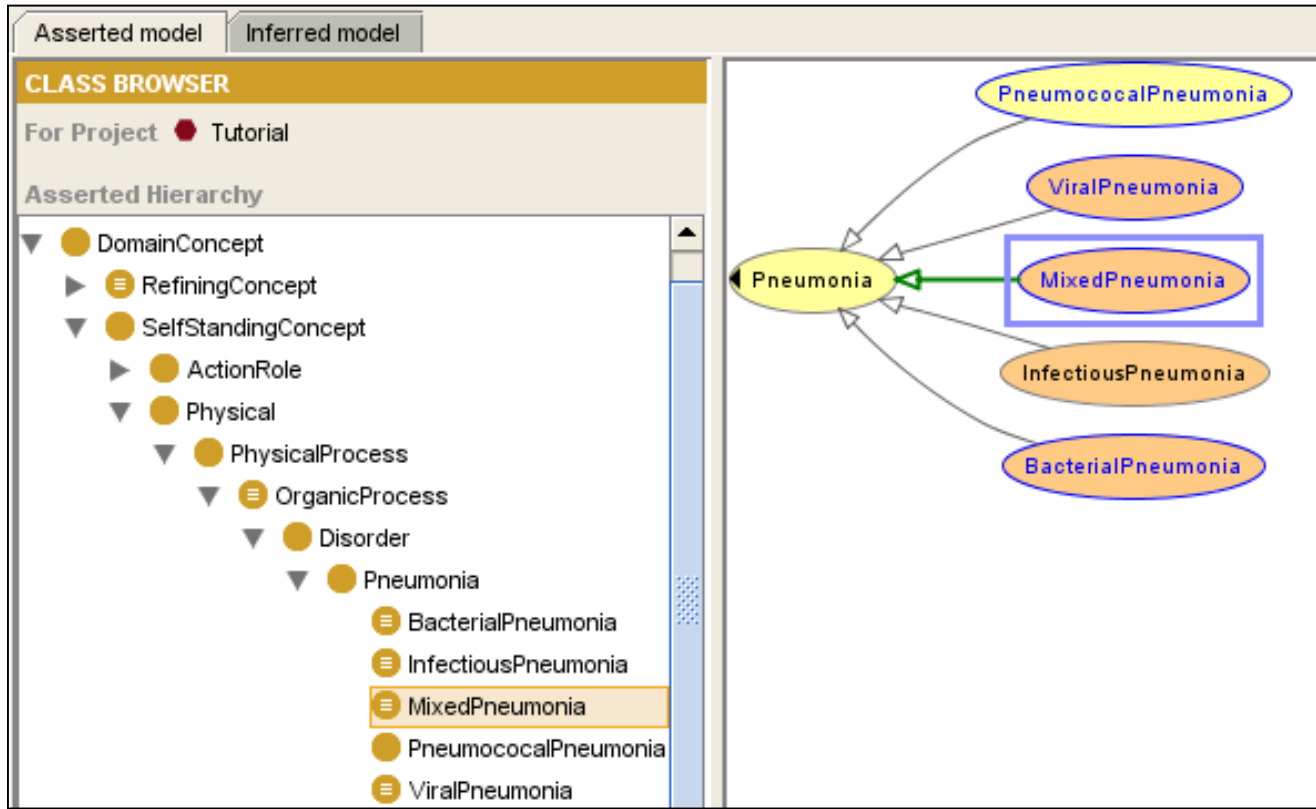
# Common errors

---

- **Lack of disjointness axioms causes what looks like "obvious misclassification"**
  - Open world assumption: classes can overlap unless you explicitly say they don't.
- **Lack of a closure axiom causes misclassification**
- **Trivial satisfaction of constraints (e.g. a missing existential 'some' declaration) can lead to weird classification**
  - e.g. an infection caused only by (herpes virus and HIV virus) will get classified as a bacterial infection!
  - ... because, the (herpes virus and HIV virus) is a null-set and the null-set is a subset of the 'bacteria' (which are disjoint with viruses) and hence the "an infection caused only by (herpes virus and HIV virus)" will be called a bacterial infection!
  - ONLY does not mean SOME

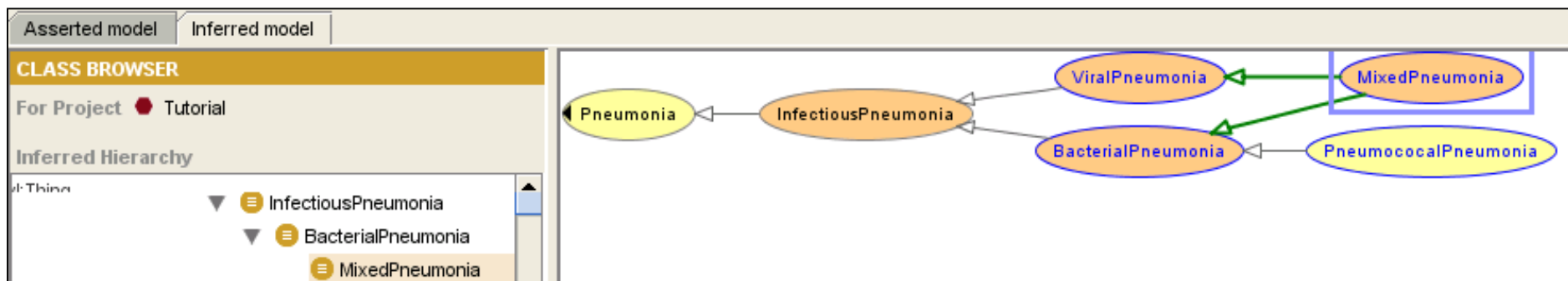
# Visualization

## Visualizing our OWL example



← Asserted Ontology

# Inferred Ontology



# Limitations

# What OWL can not do

---

- **Representation of “optional” properties is not standardized.**

Options:

- Min cardinality 0
- Member of the domain
- **No representation of uncertainty**
- **No representation of defaults and exceptions**
- **No higher order predicates**
  - OWL-DL is strictly first order
    - Can't say ... “Members of endangered species”
- **No closed world reasoning**
  - Everything must be closed explicitly
    - Take care when transforming from databases!

# Learning Materials

---

- Practical tutorial

[http://faculty.washington.edu/gennari/teaching/KR/ProtegeOWLTutorial-MattHorridge\\_v1\\_2.pdf](http://faculty.washington.edu/gennari/teaching/KR/ProtegeOWLTutorial-MattHorridge_v1_2.pdf)

- Book chapter:

Semantic Web Programming

John Hebler Matthew Fisher Ryan Blace Andrew Perez-Lopez Foreword by Mike Dean,

Principal Engineer, BBN Technologies

ISBN: 978-0-470-41801

Part II: Chapter 4: Introduction to Ontologies