OWL
(Web Ontology Language)
by example
Building an OWL ontology with Protegé

Philippe Genoud – Université Grenoble Alpes (France)
(Philippe.Genoud@imag.fr)

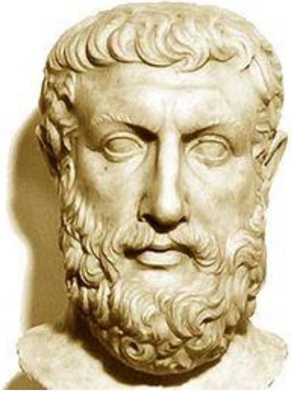
This lecture is a close adaptation of the **Matthew Horridge** tutorial :

A Practical Guide To Building OWL Ontologies
Using Protégé 4 and CO-ODE Tools Edition 1.3

http://owl.cs.manchester.ac.uk/research/co-ode/
http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf

What is an Ontology ?

- "**ontology** is the philosophical study of the nature of being, becoming, existence, or reality, as well as the basic categories of being and their relations. Traditionally listed as a part of the major branch of philosophy known as **metaphysics**, ontology deals with questions concerning what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences." http://en.wikipedia.org/wiki/Ontology



Parmenides
(c. 515 BCE - c. 460 BCE)

- "In computer science ..., an ontology is a formal framework for representing knowledge. This framework names and defines the types, properties, and interrelationships of the entities in a domain of discourse. The entities are conceptualizations (limited abstractions) of phenomena."
http://en.wikipedia.org/wiki/Ontology_%28information_science%29

An ontology is an explicit specification of a conceptualization. [...] A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose.

Thomas R. Gruber, *Towards Principles for the Design of Ontologies Used for Knowledge Sharing* in Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, 1993
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.6025&rep=rep1&type=pdf

Thomas R. Gruber
(1959 -)

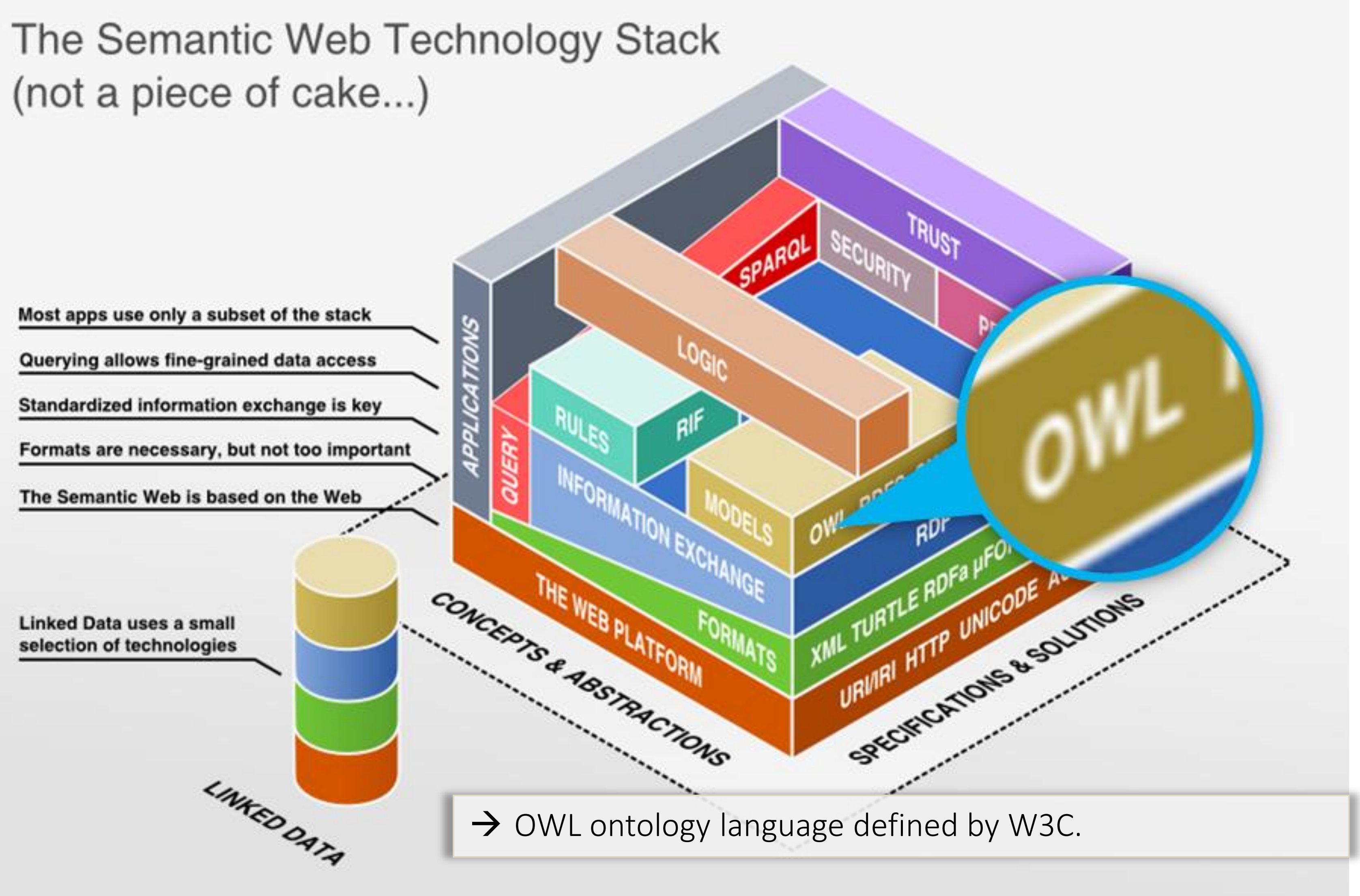
Ontology vs Vocabulary

- On the Semantic Web, vocabularies define the concepts and relationships (also referred to as "terms") used to describe and represent an area of concern. Vocabularies are used to classify the terms that can be used in a particular application, characterize possible relationships, and define possible constraints on using those terms. In practice, vocabularies can be very complex (with several thousands of terms) or very simple (describing one or two concepts only).

- There is no clear division between what is referred to as "vocabularies" and "ontologies". The trend is to use the word "ontology" for more complex, and possibly quite formal collection of terms, whereas "vocabulary" is used when such strict formalism is not necessarily used or only in a very loose sense. Vocabularies are the basic building blocks for inference techniques on the Semantic Web.

http://www.w3.org/standards/semanticweb/ontology

OWL in the Semantic Web Stack


The Semantic Web Technology Stack
(not a piece of cake...)
Most apps use only a subset of the stack
Querying allows fine-grained data access
Standardized information exchange is key
Formats are necessary, but not too important
The Semantic Web is based on the Web
Linked Data uses a small selection of technologies
LINKED DATA
APPLICATIONS
QUERY
SPARQL
SECURITY
TRUST
LOGIC
RULES
RIF
INFORMATION EXCHANGE
MODELS
OWL
RDF
FORMATS
XML TURTLE RDFa µFO
THE WEB PLATFORM
URI/IRI HTTP UNICODE
CONCEPTS & ABSTRACTIONS
SPECIFICATIONS & SOLUTIONS
OWL

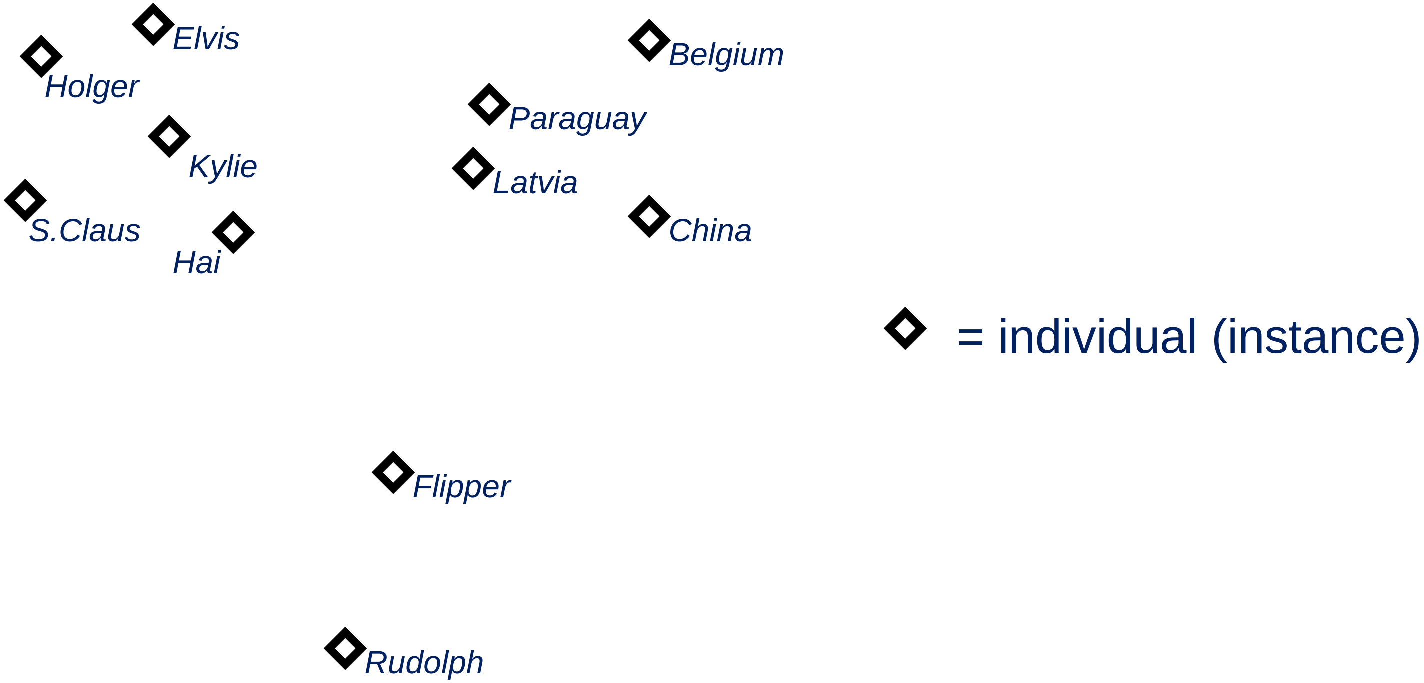
→ OWL ontology language defined by W3C.

OWL - Introduction

- OWL : **W**eb **O**ntology **L**anguage
  - a W3C standard
    - OWL 1 : W3C recommendation 10 Feb. 2004
      - http://www.w3.org/TR/owl-features/
    - OWL 2 : W3C recommendation 11 Dec. 2012
      - http://www.w3.org/TR/owl2-overview/
  - OWL vocabulary : a set of primitives described in RDF which extends the RDFS vocabulary
    - OWL namespace
      `http://www.w3.org/2002/07/owl#` ⇔ **`owl:`**

Components of OWL Ontologies

- **Individuals:** represent objects in the domain in which we are interested (the *domain of discourse*)

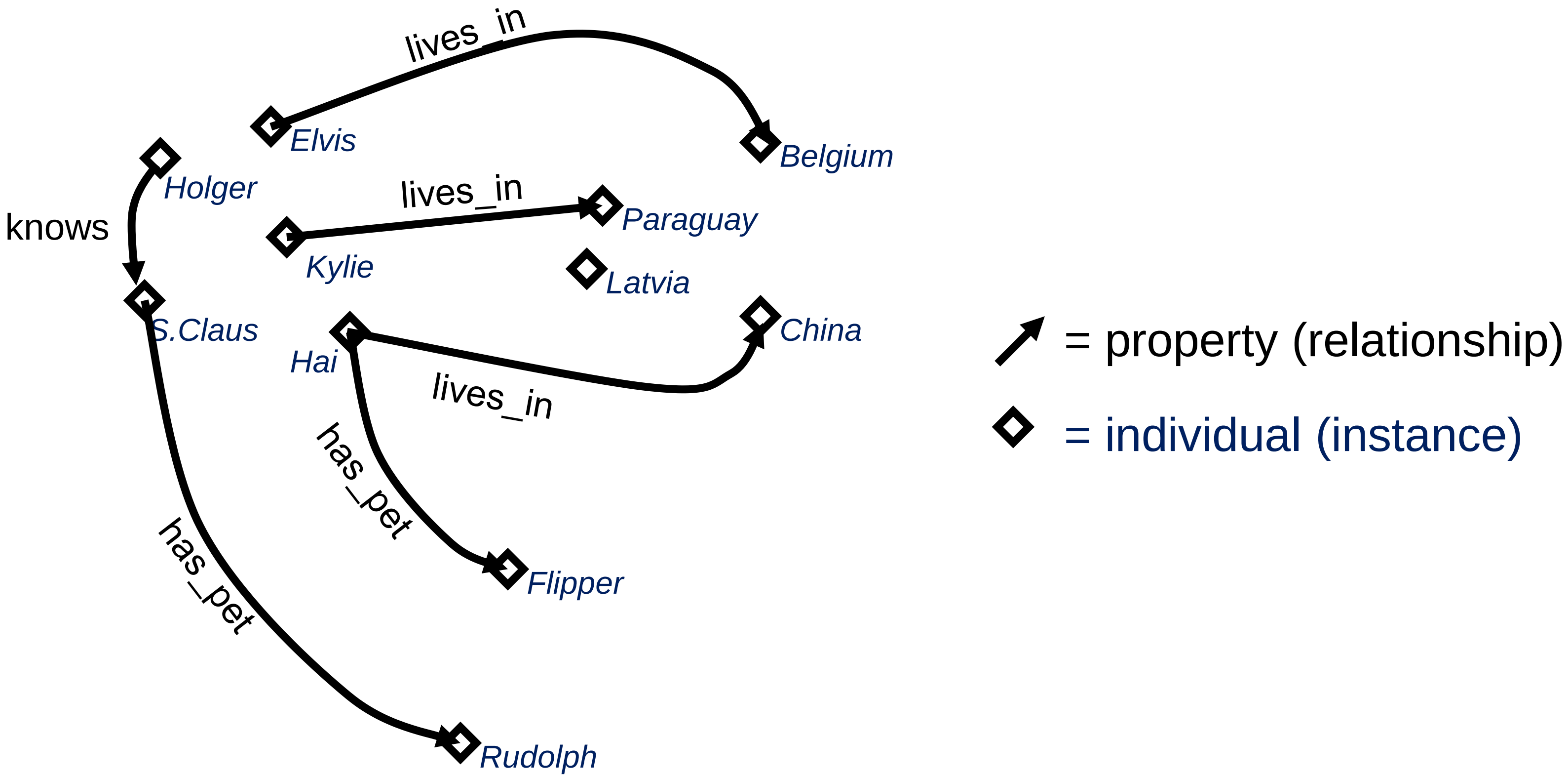


◇ = individual (instance)

- OWL does not use the Unique Name Assumption (UNA)
  - two different names (URIs) could actually refer to the same individual
  - it must be explicitly stated that individuals are the same as each other, or different to each other — otherwise they *might* be the same as each other, or they *might* be different to each other.

Components of OWL Ontologies

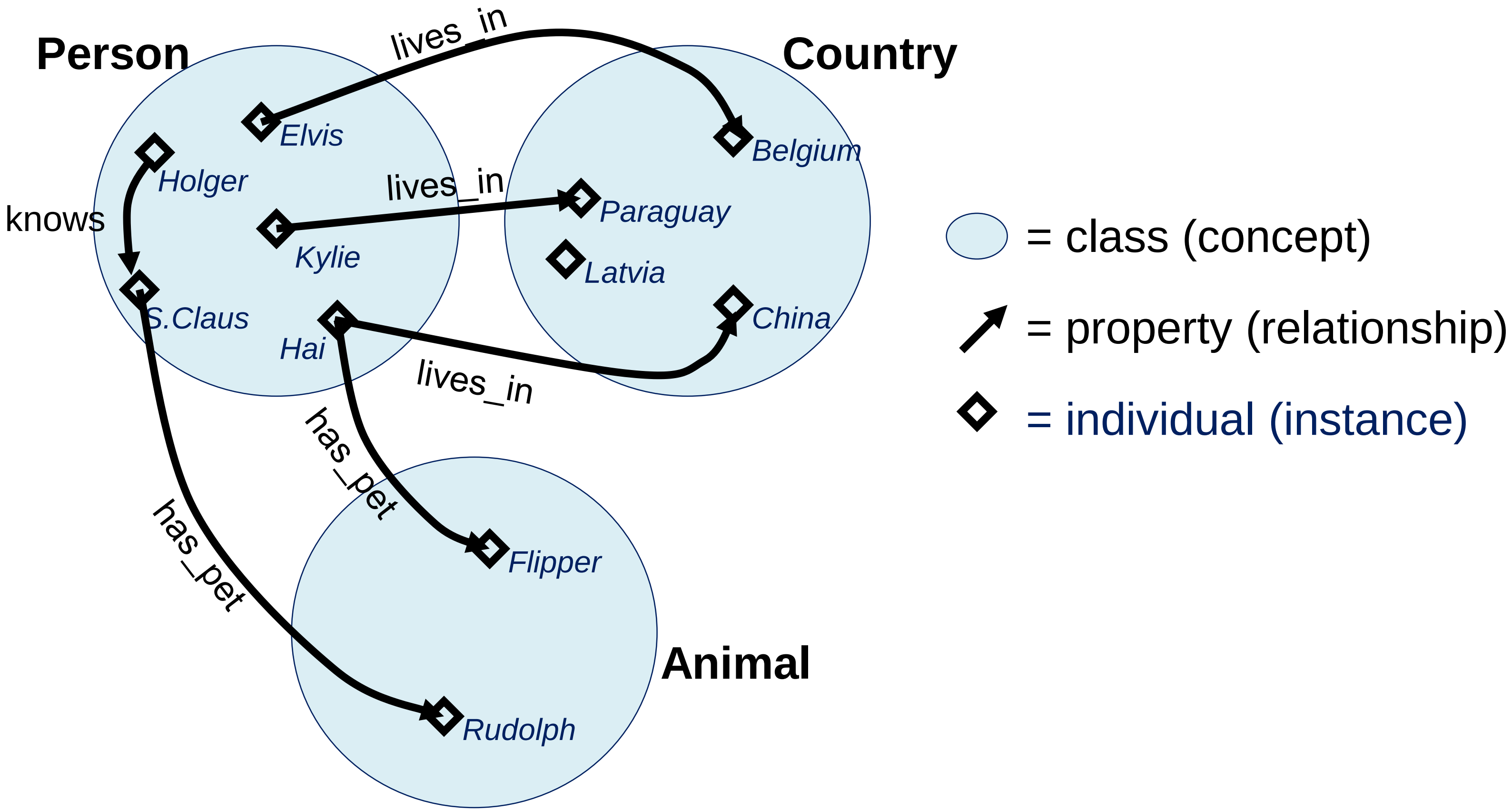
- **Properties:** binary relations on individuals, properties link two individuals together



- Properties can also link individual to literal values

Components of OWL Ontologies

- **Classes:** OWL classes are interpreted as sets that contain individuals.

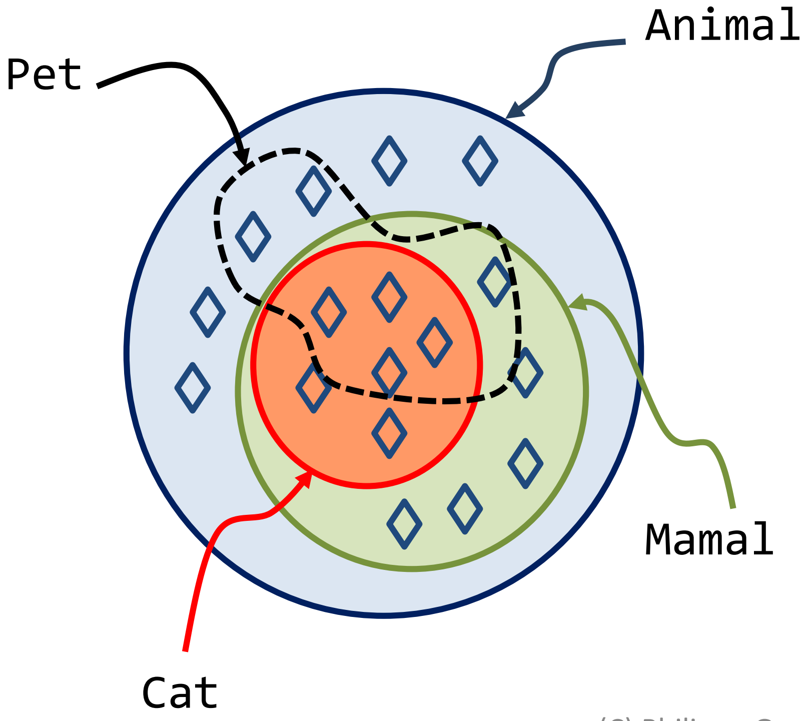


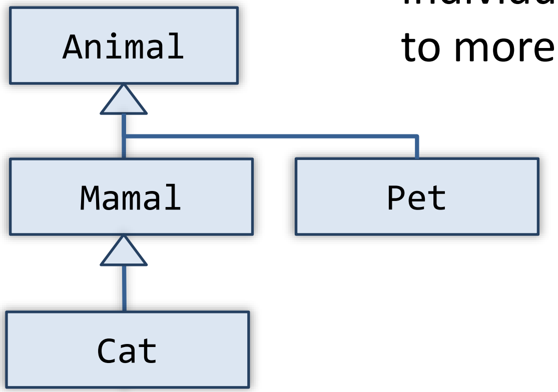
Components of OWL Ontologies

- **Classes** (continued)
  - Classes can be described using formal (mathematical) descriptions
  - Class descriptions **state precisely** the requirements for membership of the class (the conditions that must be satisfied by an individual for it to be a member of the class).
  - Different types of class descriptions
    - named classes
    - enumeration of individuals
    - union, intersection, complement of other class
    - restrictions on properties

Components of OWL Ontologies

- **Classes** (continued)
  - Classes may be organised into a superclass-subclass hierarchy (*a taxonomy*).
    - Subclasses specialise (*are subsumed by*) their superclasses.
    - *subclass* means necessary implication.
      - if A is a subclass of B then **ALL instances** of A are instances of B (without exception)





- Individuals may belong to more than one class.

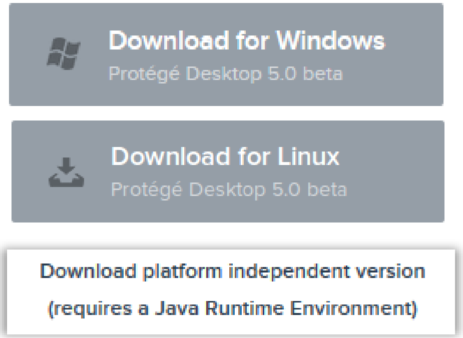
- One of the key features of OWL-DL is that these superclass-subclass relationships can be computed automatically (inferred) by a *reasoner*

Protégé

- A knowledge modelling environment
- A free, open source software
- Developed by Stanford / Manchester
- Has a large user community (approx. 240k)
- Protégé 4+ built solely on OWL modelling language
- 2 versions:
  - Desktop application based on Eclipse RCP
    Supports development of plugins to allow backend / interface extensions
  - Web application (Web Protégé)

http://protege.stanford.edu

Download and install Protégé Desktop on your computer



Download for Windows
Protégé Desktop 5.0 beta

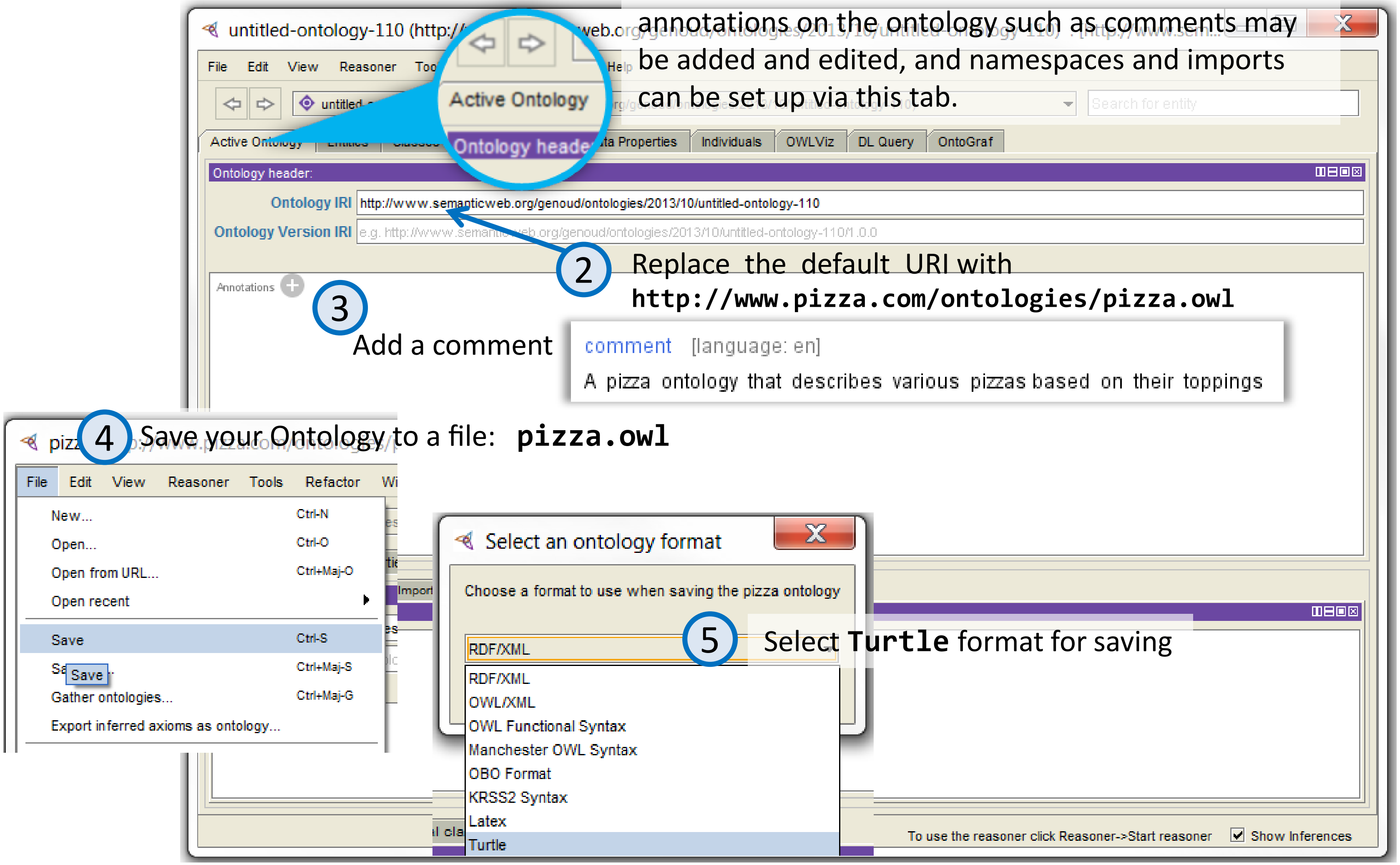
Download for Linux
Protégé Desktop 5.0 beta

Download platform independent version
(requires a Java Runtime Environment)

Creating a new OWL Ontology

1 Start Protégé

allows information about the ontology to be specified. For example, the ontology URI can be changed, annotations on the ontology such as comments may be added and edited, and namespaces and imports can be set up via this tab.



Ontology header:

Ontology IRI: http://www.semanticweb.org/genoud/ontologies/2013/10/untitled-ontology-110

Ontology Version IRI: e.g. http://www.semanticweb.org/genoud/ontologies/2013/10/untitled-ontology-110/1.0.0

2 Replace the default URI with **`http://www.pizza.com/ontologies/pizza.owl`**

3 Add a comment

comment [language: en]
A pizza ontology that describes various pizzas based on their toppings

4 Save your Ontology to a file: **`pizza.owl`**

pizza (http://www.pizza.com/ontologies/
File
Edit
View
Reasoner
Tools
Refactor
New... Ctrl-N
Open... Ctrl-O
Open from URL... Ctrl+Maj-O
Open recent
Save Ctrl-S
Save as... Ctrl+Maj-S
Gather ontologies... Ctrl+Maj-G
Export inferred axioms as ontology...

Select an ontology format

Choose a format to use when saving the pizza ontology

RDF/XML

- RDF/XML
- OWL/XML
- OWL Functional Syntax
- Manchester OWL Syntax
- OBO Format
- KRSS2 Syntax
- Latex
- Turtle

5 Select **`Turtle`** format for saving

owl:Ontology

RDF/XML

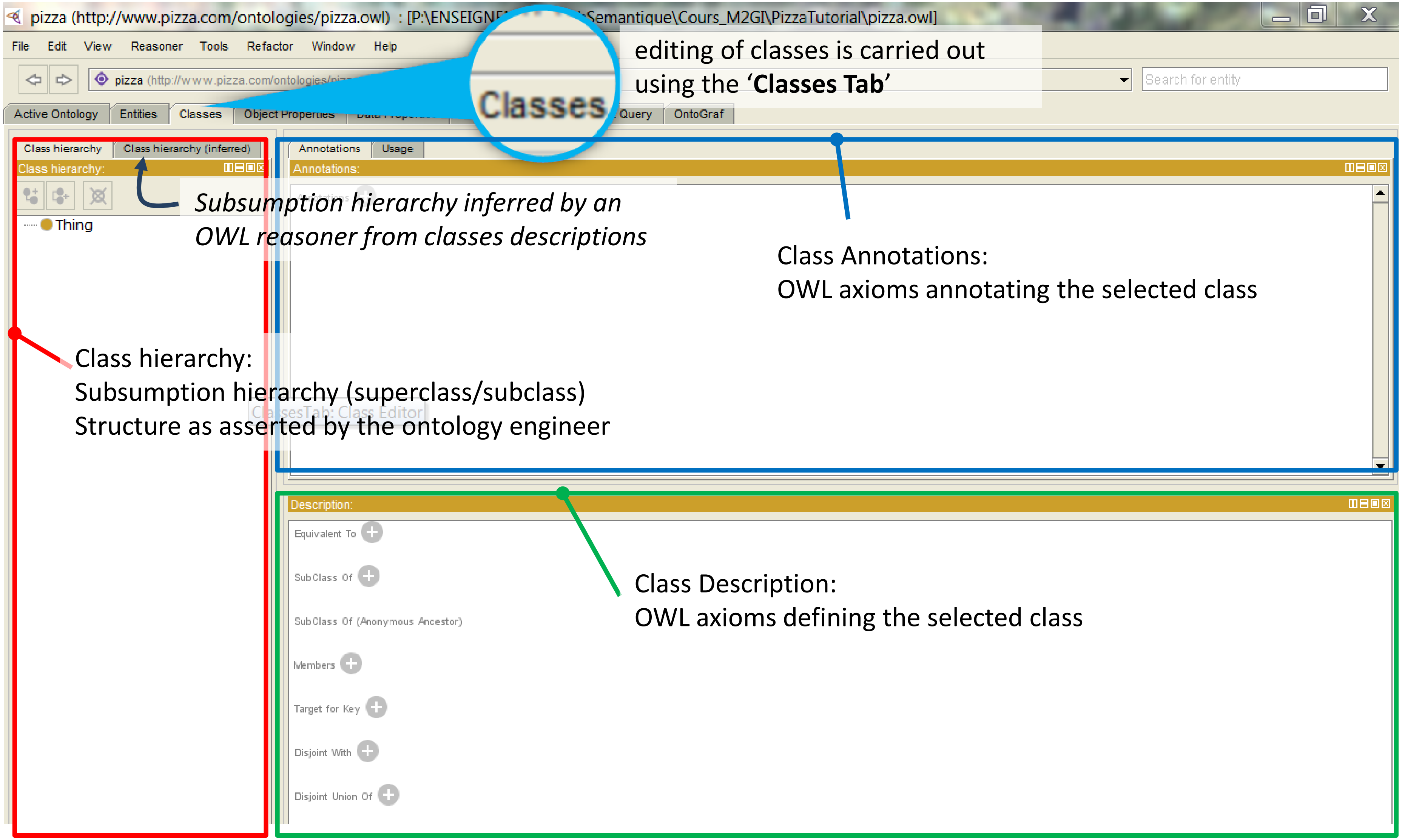
```
<?xml version="1.0"?>

<rdf:RDF xmlns="http://www.pizza.com/ontologies#"
     xml:base="http://www.pizza.com/ontologies"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about="http://www.pizza.com/ontologies">
        <rdfs:comment> A  pizza  ontology  that  describes  various  pizzas
           based  on  their  toppings.
        </rdfs:comment>
    </owl:Ontology>
</rdf:RDF>
```

Turtle

All resources defined in this ontology will be identified by hash URI beginning with this prefix

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

<http://www.pizza.com/ontologies> rdf:type owl:Ontology ;
                                  rdfs:comment "A  pizza  ontology  that  describes  various  pizzas
                                                   based  on  their  toppings." .

###  Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net
```

ClassesTab: Class Editor



editing of classes is carried out using the ‘**Classes Tab**’

Subsumption hierarchy inferred by an OWL reasoner from classes descriptions

Class Annotations:
OWL axioms annotating the selected class

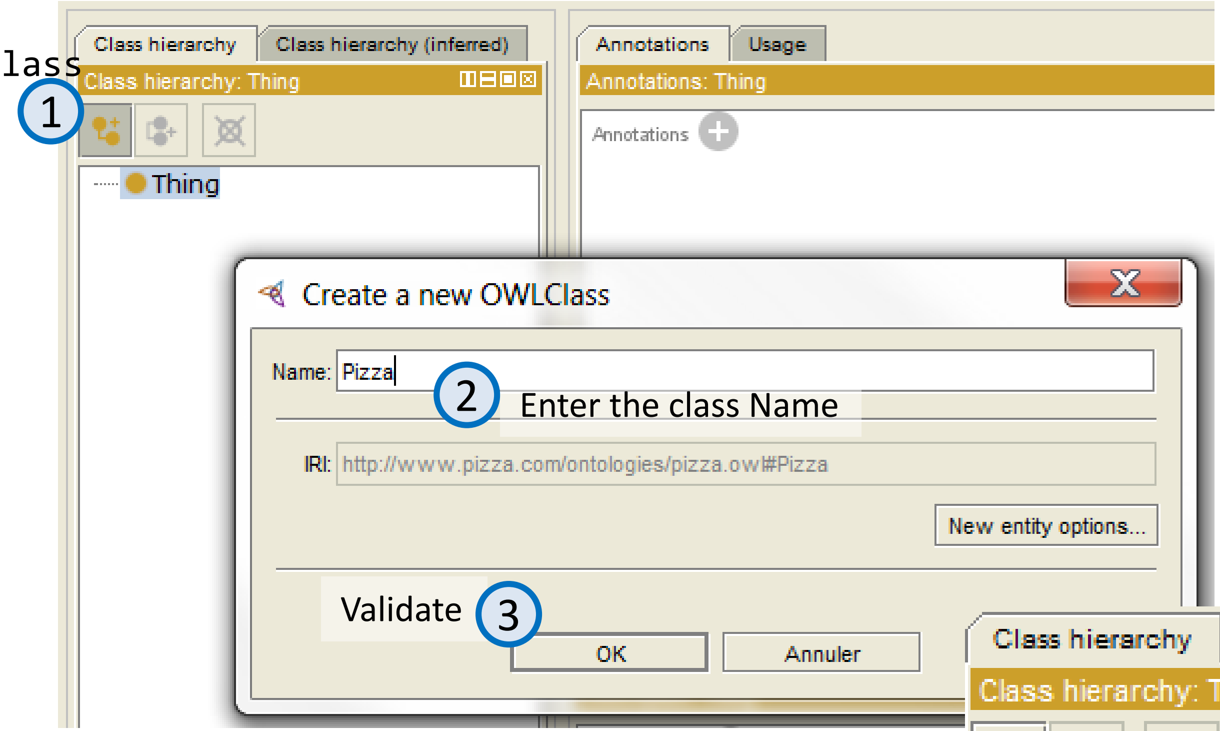
Class hierarchy:
Subsumption hierarchy (superclass/subclass)
Structure as asserted by the ontology engineer

Class Description:
OWL axioms defining the selected class

Creating classes

Create classes **Pizza**, **PizzaTopping** and **PizzaBase** as subclasses of **Thing**

1 Press the `Add Subclass` button



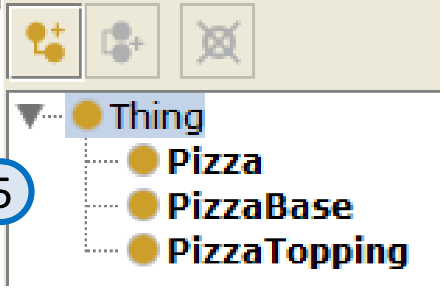
Name: Pizza

IRI: http://www.pizza.com/ontologies/pizza.owl#Pizza

2 Enter the class Name

3 Validate

4 Repeat to create PizzaTopping and PizzaBase (try to use the `Add Sibling Class` button)

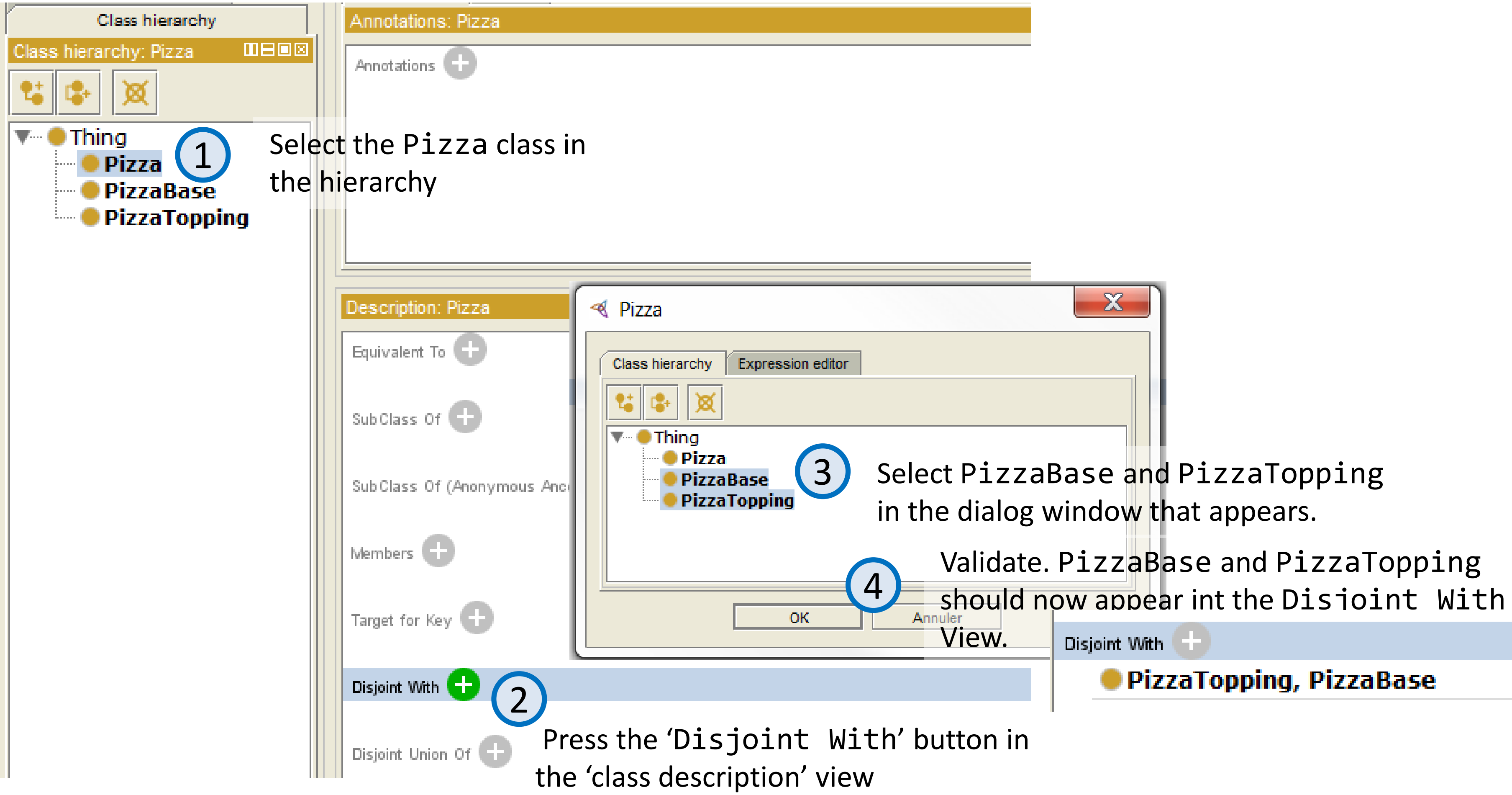


Class hierarchy | Class hierar
Class hierarchy: Thing

5 Ensure you have this initial Class Hierarchy (taxonomy)

Disjoint classes

Let's say the `Pizza`, `PizzaBase` and `PizzaTopping` classes are **disjoint**
→ an individual (or object) cannot be an instance of more than one of these three classes



1 Select the `Pizza` class in the hierarchy

2 Press the '`Disjoint With`' button in the 'class description' view

3 Select `PizzaBase` and `PizzaTopping` in the dialog window that appears.

4 Validate. `PizzaBase` and `PizzaTopping` should now appear int the `Disjoint With` View.

Disjoint classes

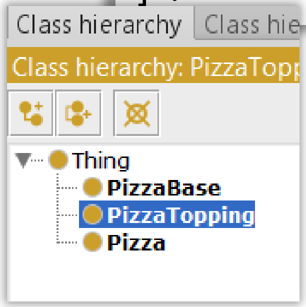
Turtle

```
...
#################################################################
#    Classes
#################################################################
###  http://www.pizza.com/ontologies/pizza.owl#Pizza
:Pizza rdf:type owl:Class .

###  http://www.pizza.com/ontologies/pizza.owl#PizzaBase
:PizzaBase rdf:type owl:Class .

###  http://www.pizza.com/ontologies/pizza.owl#PizzaTopping
:PizzaTopping rdf:type owl:Class .

#################################################################
#    General axioms
#################################################################
[ rdf:type owl:AllDisjointClasses ;
  owl:members ( :Pizza
                :PizzaBase
                :PizzaTopping )
] .
```

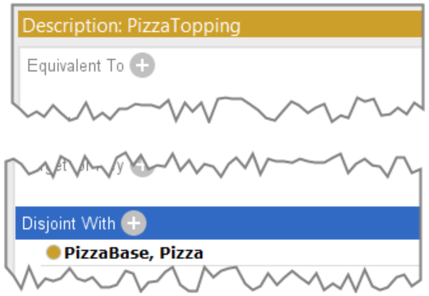
Blank node

n-ary axiom
https://www.w3.org/2007/OWL/wiki/FullSemanticsNaryAxioms

RDF list

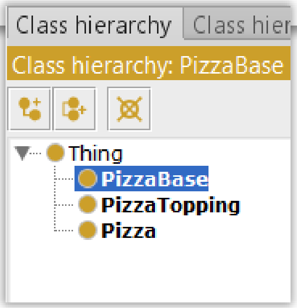


Thing
- PizzaBase
- PizzaTopping
- Pizza

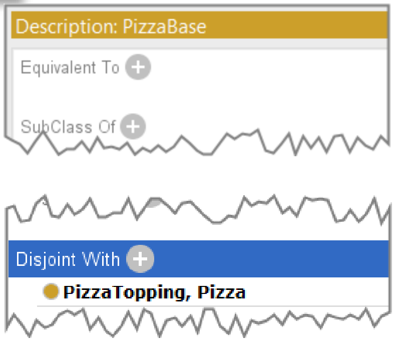

Description: PizzaTopping

Equivalent To

Disjoint With
- PizzaBase, Pizza



Thing
- PizzaBase
- PizzaTopping
- Pizza


Description: PizzaBase

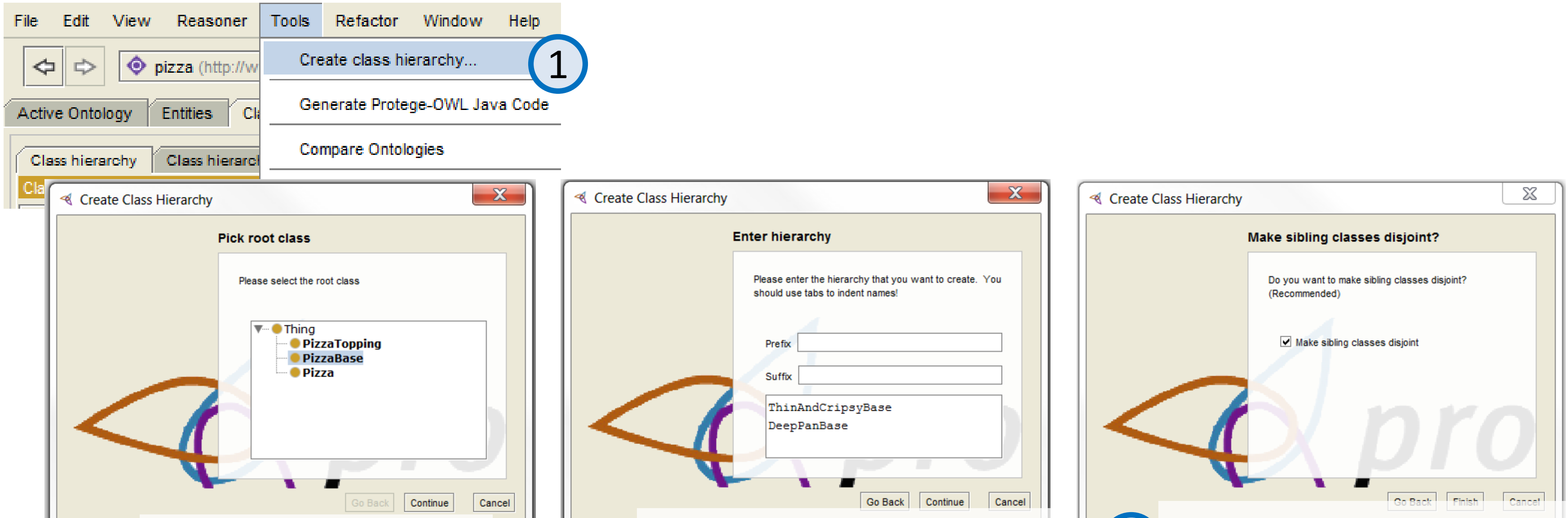
Equivalent To

SubClass Of

Disjoint With
- PizzaTopping, Pizza

Create a Class Hierarchy

Create **ThinAndCripsyBase** and **DeepPanBase** as subclasses of **PizzaBase**



Pick root class

Please select the root class

- Thing
  - PizzaTopping
  - PizzaBase
  - Pizza

Enter hierarchy

Please enter the hierarchy that you want to create. You should use tabs to indent names!

Prefix

Suffix

```
ThinAndCripsyBase
DeepPanBase
```

Make sibling classes disjoint?

Do you want to make sibling classes disjoint? (Recommended)

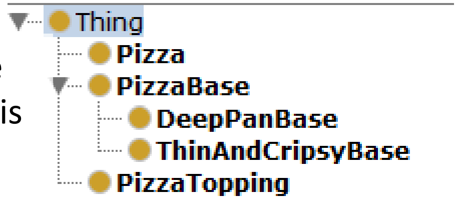
☑ Make sibling classes disjoint

2 Select the PizzaBase as the root class

3 Type in the names of the classes to create

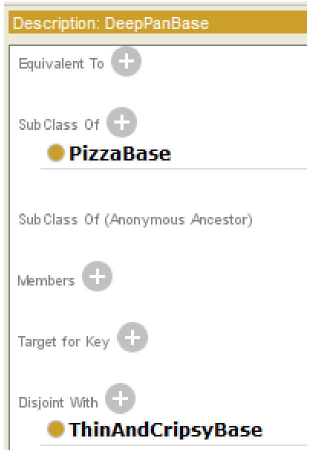
4 Make the new classes disjoint

5 Ensure that the class hierarchy is correct



- Thing
  - Pizza
  - PizzaBase
    - DeepPanBase
    - ThinAndCripsyBase
  - PizzaTopping

6 Ensure that DeepPanBase and ThinAndCripsyBase classes have correct descriptions



Description: DeepPanBase

Equivalent To

SubClass Of
PizzaBase

SubClass Of (Anonymous Ancestor)

Members

Target for Key

Disjoint With
ThinAndCripsyBase

Create a Class Hierarchy (continued)

Turtle

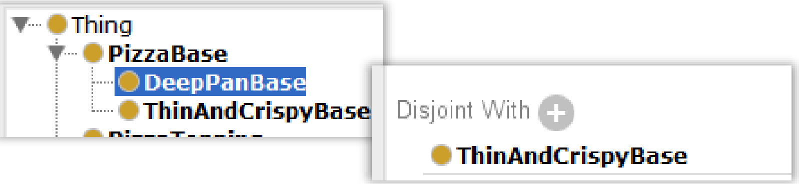
```
...
#################################################################
#    Classes
#################################################################

###  http://www.pizza.com/ontologies/pizza.owl# DeepPanBase
:DeepPanBase rdf:type owl:Class ;
             rdfs:subClassOf :PizzaBase ;
             owl:disjointWith :ThinAndCrispyBase .

###  http://www.pizza.com/ontologies/pizza.owl#Pizza
:Pizza rdf:type owl:Class .

###  http://www.pizza.com/ontologies/pizza.owl#PizzaBase
:PizzaBase rdf:type owl:Class .


###  http://www.pizza.com/ontologies/pizza.owl#PizzaTopping
:PizzaTopping rdf:type owl:Class .


###  http://www.pizza.com/ontologies/pizza.owl#ThinAndCrispyBase
:ThinAndCrispyBase rdf:type owl:Class ;
                   rdfs:subClassOf :PizzaBase .
```

Declarations order doesn’t matters

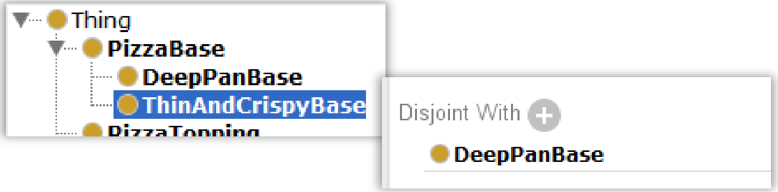
binary axiom
OWL semantics implies that it’s a symmetric property

inferred

```
:ThinAndCrispyBase
    owl:disjointWith :DeepPanBase .
```


```
Thing
  PizzaBase
    DeepPanBase
    ThinAndCrispyBase
  PizzaTopping
```
Disjoint With

ThinAndCrispyBase


```
Thing
  PizzaBase
    DeepPanBase
    ThinAndCrispyBase
  PizzaTopping
```
Disjoint With

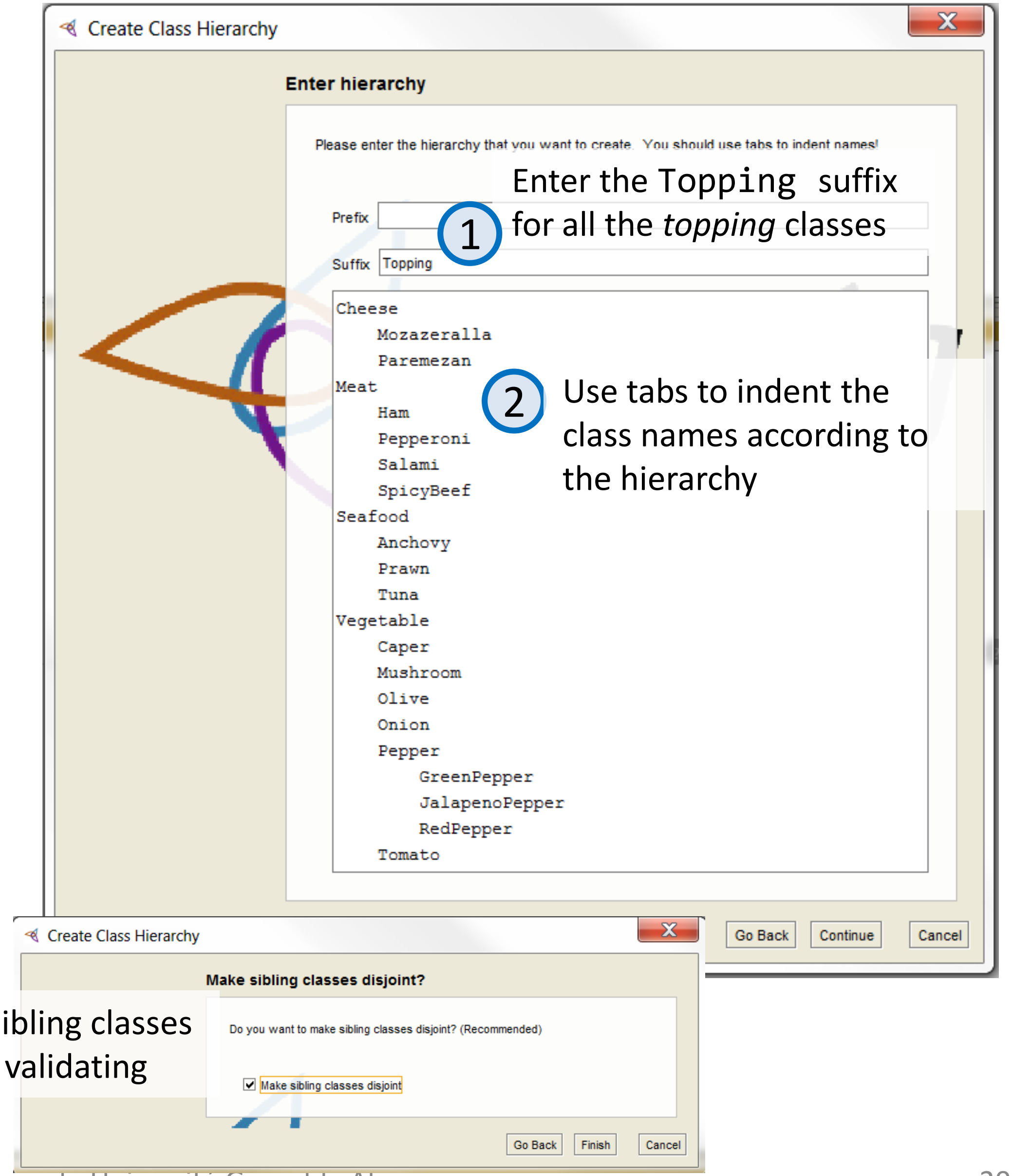
DeepPanBase

Create a Class Hierarchy (continued)

Create some subclasses of `PizzaTopping` :
`CheeseTopping`, `MeatTopping`, ...

Hierarchy to create (without the Topping suffix)

```
Cheese
    Mozarella
    Paremezan
Meat
    Ham
    Pepperoni
    Salami
    SpicyBeef
Seafood
    Anchovy
    Prawn
    Tuna
Vegetable
    Caper
    Mushroom
    Olive
    Onion
    Pepper
        GreenPepper
        JalapenoPepper
        RedPepper
    Tomato
```



Create Class Hierarchy

Enter hierarchy

Please enter the hierarchy that you want to create. You should use tabs to indent names!

Prefix:

Suffix: Topping

```
Cheese
    Mozazeralla
    Paremezan
Meat
    Ham
    Pepperoni
    Salami
    SpicyBeef
Seafood
    Anchovy
    Prawn
    Tuna
Vegetable
    Caper
    Mushroom
    Olive
    Onion
    Pepper
        GreenPepper
        JalapenoPepper
        RedPepper
    Tomato
```

1 Enter the `Topping` suffix for all the *topping* classes

2 Use tabs to indent the class names according to the hierarchy

Create Class Hierarchy

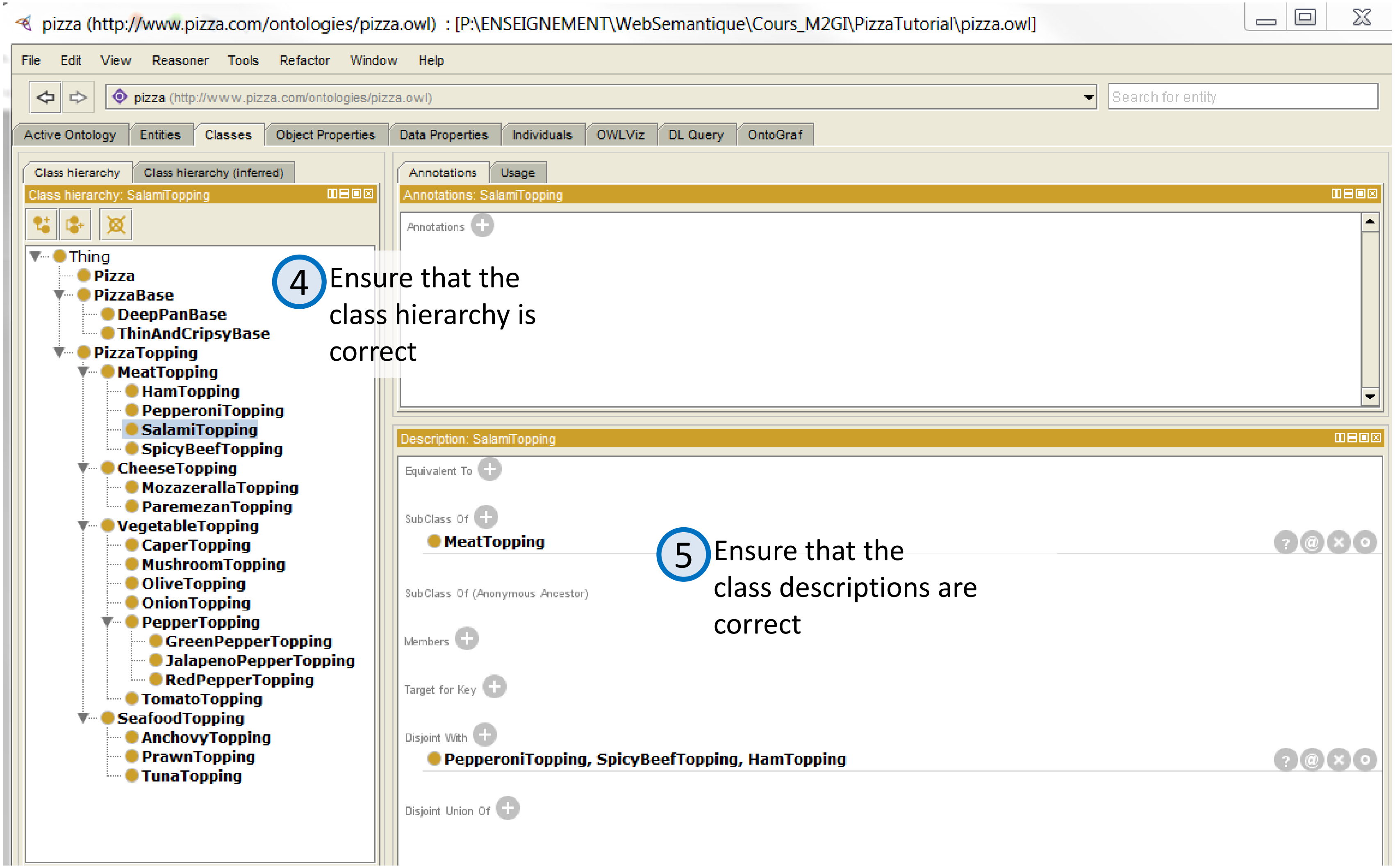
Make sibling classes disjoint?

Do you want to make sibling classes disjoint? (Recommended)

☑ Make sibling classes disjoint

3 Make all the sibling classes disjoint when validating

Creating a Class Hierarchy (continued)



pizza (http://www.pizza.com/ontologies/pizza.owl) : [P:\ENSEIGNEMENT\WebSemantique\Cours_M2GI\PizzaTutorial\pizza.owl]

Class hierarchy: SalamiTopping

- Thing
  - Pizza
  - PizzaBase
    - DeepPanBase
    - ThinAndCripsyBase
  - PizzaTopping
    - MeatTopping
      - HamTopping
      - PepperoniTopping
      - SalamiTopping
      - SpicyBeefTopping
    - CheeseTopping
      - MozazerallaTopping
      - ParemezanTopping
    - VegetableTopping
      - CaperTopping
      - MushroomTopping
      - OliveTopping
      - OnionTopping
      - PepperTopping
        - GreenPepperTopping
        - JalapenoPepperTopping
        - RedPepperTopping
      - TomatoTopping
    - SeafoodTopping
      - AnchovyTopping
      - PrawnTopping
      - TunaTopping

Annotations: SalamiTopping

Annotations

Description: SalamiTopping

Equivalent To

SubClass Of
- MeatTopping

SubClass Of (Anonymous Ancestor)

Members

Target for Key

Disjoint With
- PepperoniTopping, SpicyBeefTopping, HamTopping

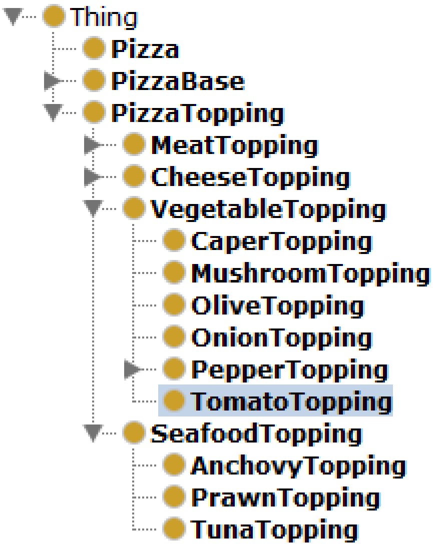
Disjoint Union Of

4 Ensure that the class hierarchy is correct

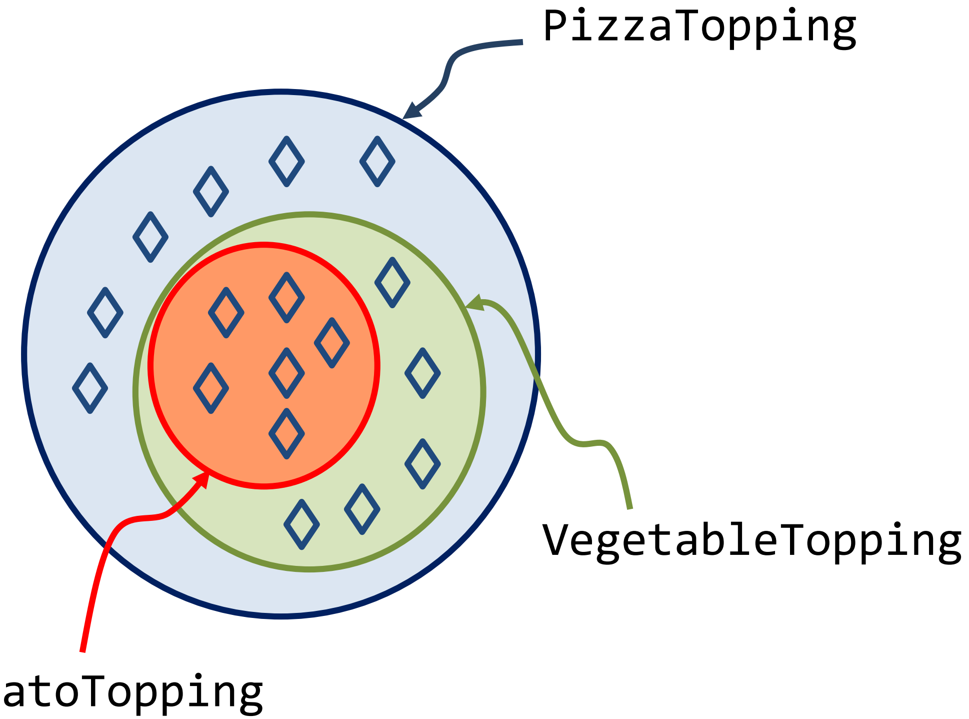
5 Ensure that the class descriptions are correct

Class Hierarchy

- In OWL *subclass* means necessary implication.
  - if A is a subclass of B then **ALL instances** of A are instances of B (without exception)

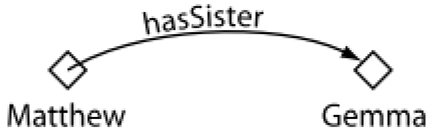


- Thing
  - Pizza
  - PizzaBase
  - PizzaTopping
    - MeatTopping
    - CheeseTopping
    - VegetableTopping
      - CaperTopping
      - MushroomTopping
      - OliveTopping
      - OnionTopping
      - PepperTopping
      - TomatoTopping
    - SeafoodTopping
      - AnchovyTopping
      - PrawnTopping
      - TunaTopping



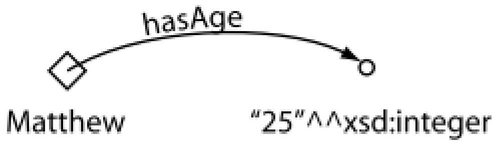
OWL Properties

- OWL Properties represent relationships
- two main types of properties
  - **Object properties** : relationships between two individuals

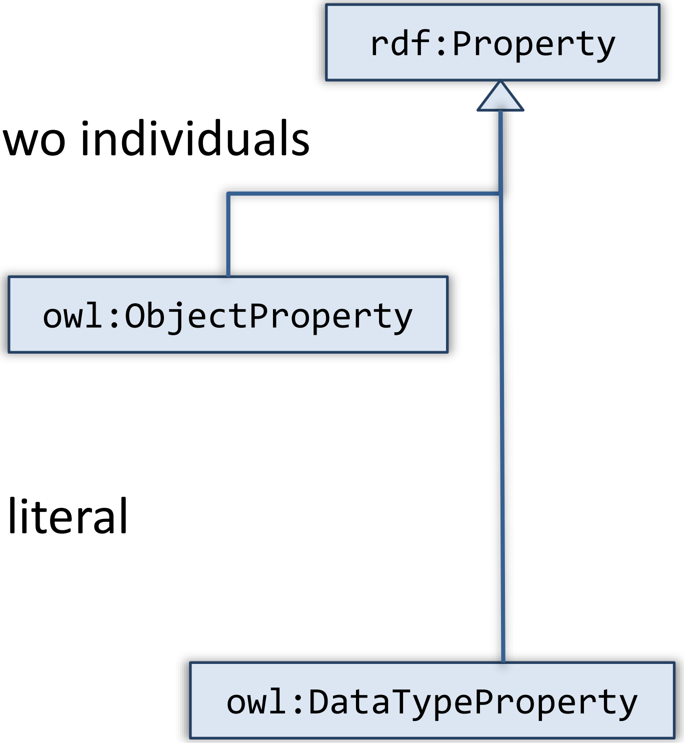


An object property linking the individual Matthew to the individual Gemma

  - **Datatype properties** : link an individual to a literal

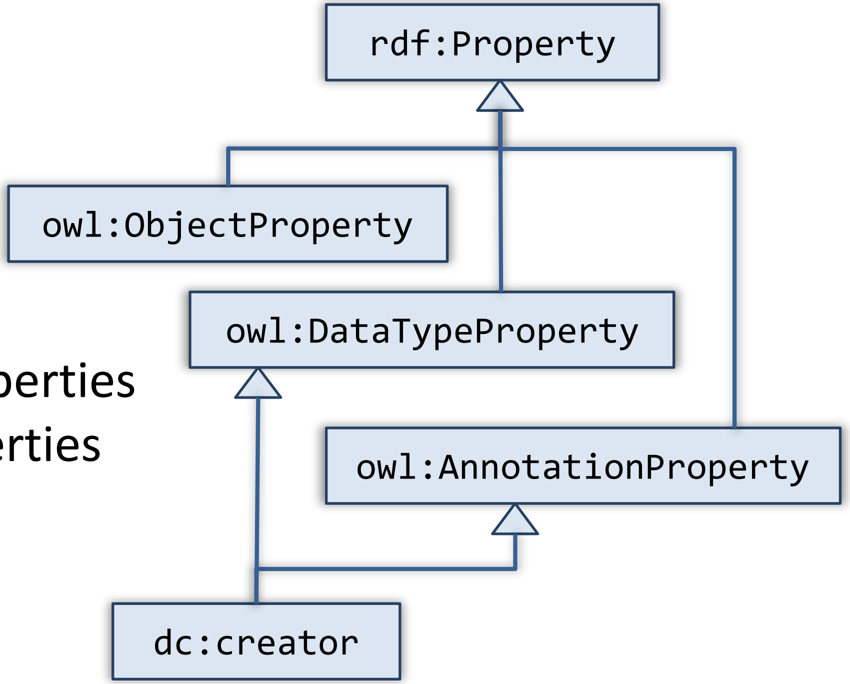


A datatype property linking the individual Matthew to the data literal '25', which has a type of an xsd:integer.



OWL properties

- a third type of property
  - **Annotation properties:** can be used to add information (metadata - data about data) to classes, individuals and object/datatype properties.
  - Object properties and Datatype properties may be marked as Annotation properties

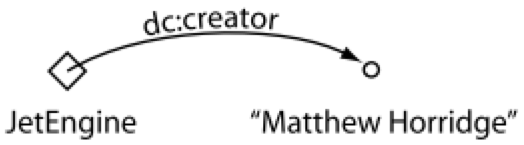

rdf:Property

owl:ObjectProperty

owl:DataTypeProperty

owl:AnnotationProperty

dc:creator

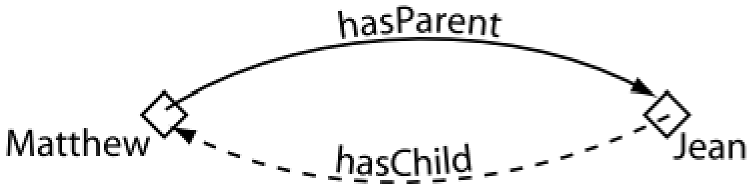


An annotation property, linking the class 'JetEngine' to the data literal (string) "Matthew Horridge".

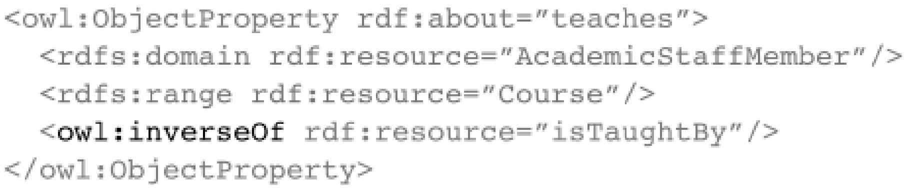
Inverse properties

- Each object property may have a corresponding inverse property.
  - If some property links individual **a** to individual **b** then its inverse property will link individual **b** to individual **a**.

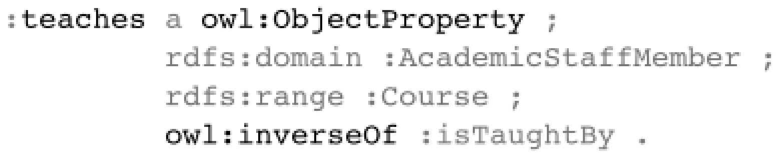
Exemples



hasParent has an inverse property that is hasChild



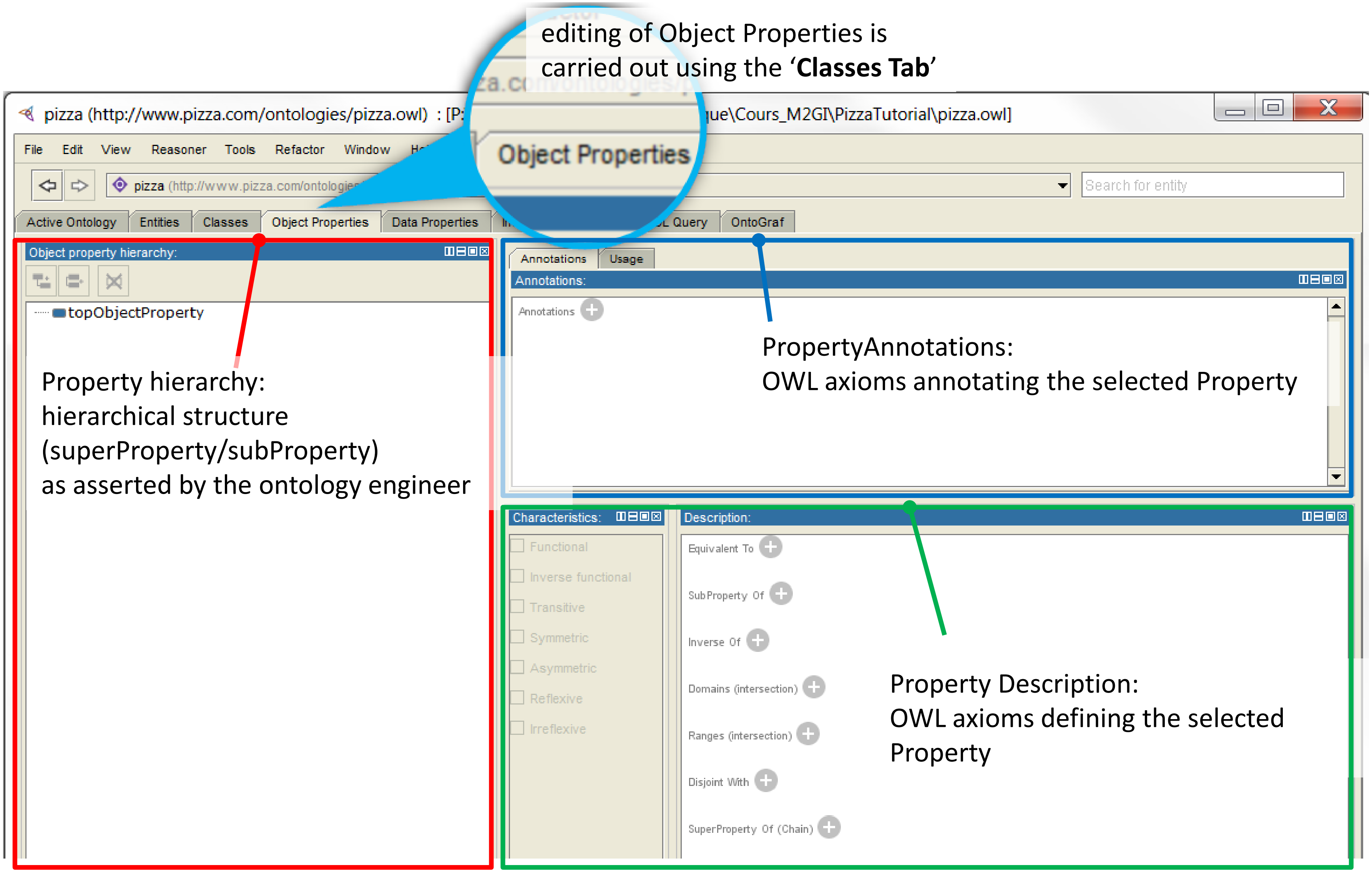
```
<owl:ObjectProperty rdf:about="teaches">
  <rdfs:domain rdf:resource="AcademicStaffMember"/>
  <rdfs:range rdf:resource="Course"/>
  <owl:inverseOf rdf:resource="isTaughtBy"/>
</owl:ObjectProperty>
```



```
:teaches a owl:ObjectProperty ;
         rdfs:domain :AcademicStaffMember ;
         rdfs:range :Course ;
         owl:inverseOf :isTaughtBy .
```

Object Properties Tab

editing of Object Properties is carried out using the '**Classes Tab**'



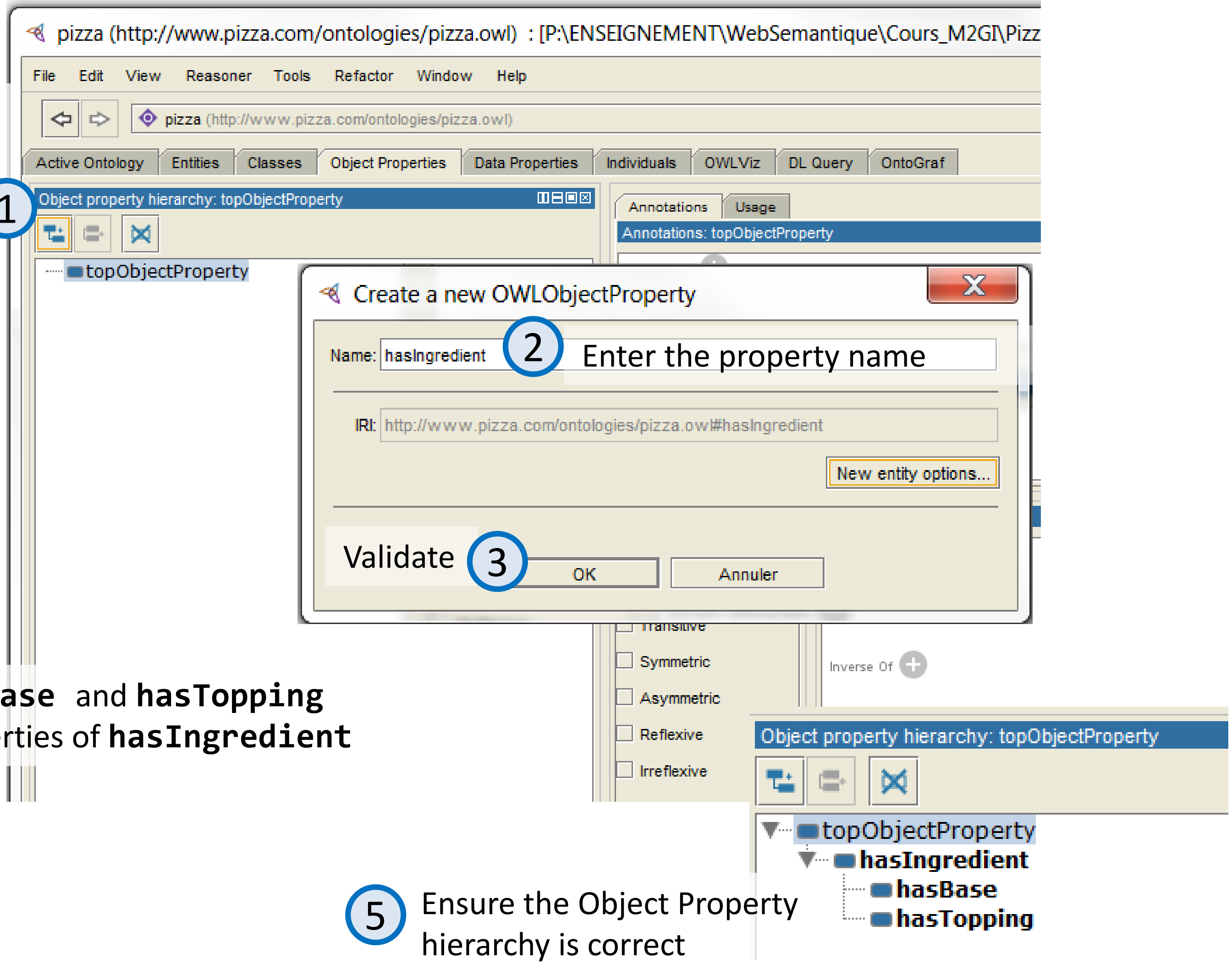
Property hierarchy:
hierarchical structure
(superProperty/subProperty)
as asserted by the ontology engineer

PropertyAnnotations:
OWL axioms annotating the selected Property

Property Description:
OWL axioms defining the selected Property

Create an Object Property hierarchy

Create an Object Property **`hasIngredient`** as subProperty of **`topObjectProperty`**


pizza (http://www.pizza.com/ontologies/pizza.owl) : [P:\ENSEIGNEMENT\WebSemantique\Cours_M2GI\Pizz
File Edit View Reasoner Tools Refactor Window Help
pizza (http://www.pizza.com/ontologies/pizza.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Object property hierarchy: topObjectProperty
topObjectProperty
Annotations Usage
Annotations: topObjectProperty
Create a new OWLObjectProperty
New entity options...
OK
Annuler
Transitive
□ Symmetric
□ Asymmetric
□ Reflexive
□ Irreflexive
Inverse Of
Object property hierarchy: topObjectProperty
topObjectProperty
hasIngredient
hasBase
hasTopping

Name: hasIngredient

IRI: http://www.pizza.com/ontologies/pizza.owl#hasIngredient

1 Press the Add `subproperty` button

2 Enter the property name

3 Validate

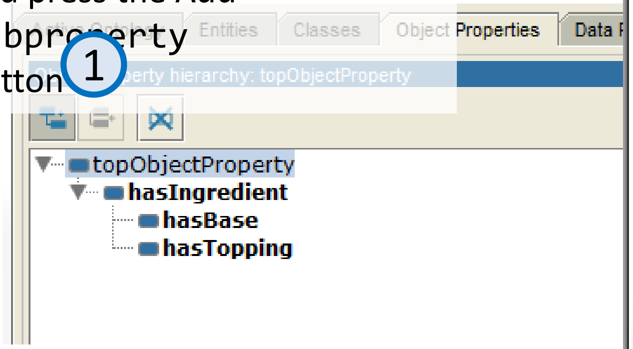
4 Create **`hasBase`** and **`hasTopping`** as sub properties of **`hasIngredient`**

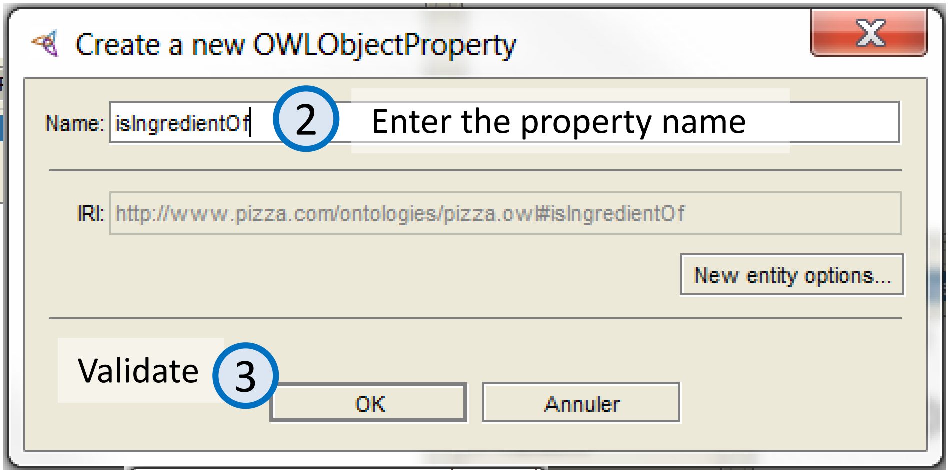
5 Ensure the Object Property hierarchy is correct

Create inverse properties

Create an Object Property **isIngredientOf** as the inverse of **hasIngredient**

Select **topObjectProperty** and press the Add subproperty button ①



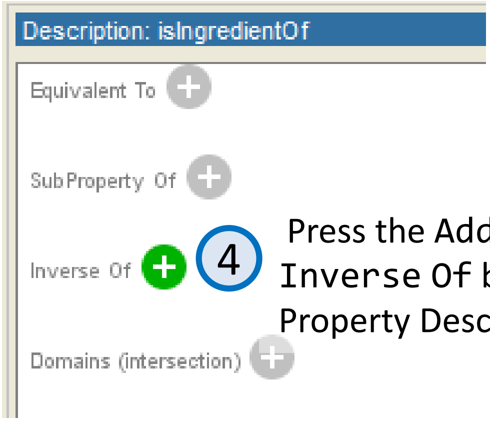


Name: isIngredientOf

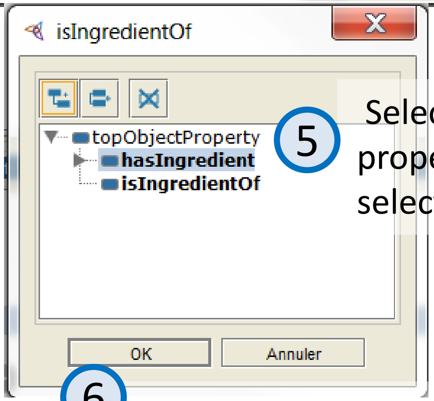
IRI: http://www.pizza.com/ontologies/pizza.owl#isIngredientOf

② Enter the property name

Validate ③

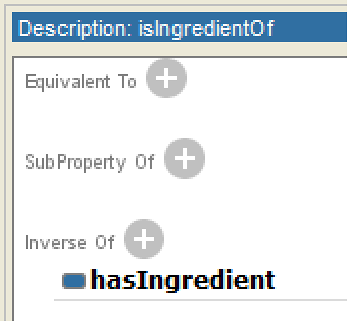


④ Press the Add icon next to Inverse Of button on the Property Description view



⑤ Select the **hasIngredient** property in the property selection dialog

⑥ Validate and ensure that **isIngredientOf** has a correct description



Property hierarchy

Turtle

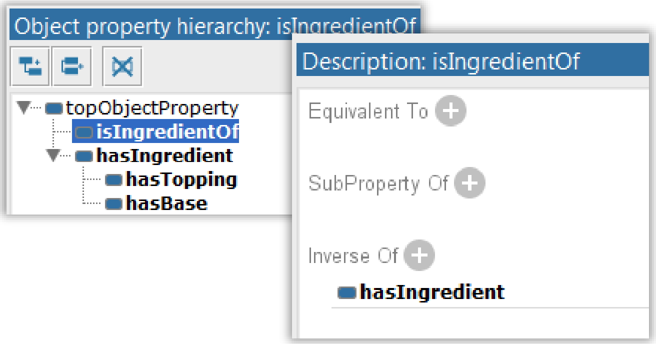
```
...
##################################################################
#    Object Properties
##################################################################
###  http://www.pizza.com/ontologies/pizza.owl#hasBase
:hasBase rdf:type owl:ObjectProperty ;
         rdfs:subPropertyOf :hasIngredient .


###  http://www.pizza.com/ontologies/pizza.owl#hasIngredient
:hasIngredient rdf:type owl:ObjectProperty .


###  http://www.pizza.com/ontologies/pizza.owl#hasTopping
:hasTopping rdf:type owl:ObjectProperty ;
            rdfs:subPropertyOf :hasIngredient .


###  http://www.pizza.com/ontologies/pizza.owl#isIngredientOf
:isIngredientOf rdf:type owl:ObjectProperty ;
                owl:inverseOf :hasIngredient .
```

Declarations order doesn't matters

binary axiom
OWL semantics implies that it's a symmetric property

inferred

```
:hasIngredient
    owl:inverseOf :isIngredient .
```



Object property hierarchy: isIngredientOf

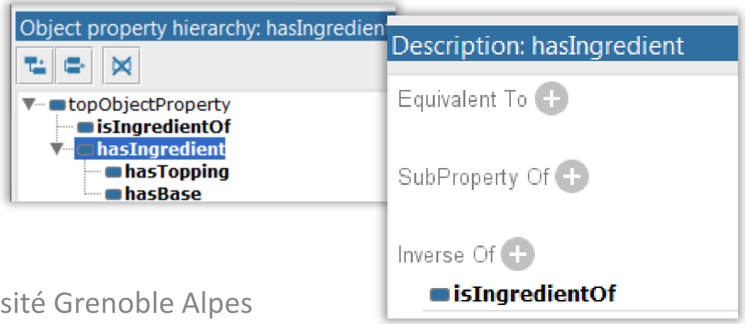
- topObjectProperty
  - isIngredientOf
  - hasIngredient
    - hasTopping
    - hasBase

Description: isIngredientOf

Equivalent To

SubProperty Of

Inverse Of
- hasIngredient



Object property hierarchy: hasIngredien

- topObjectProperty
  - isIngredientOf
  - hasIngredient
    - hasTopping
    - hasBase

Description: hasIngredient

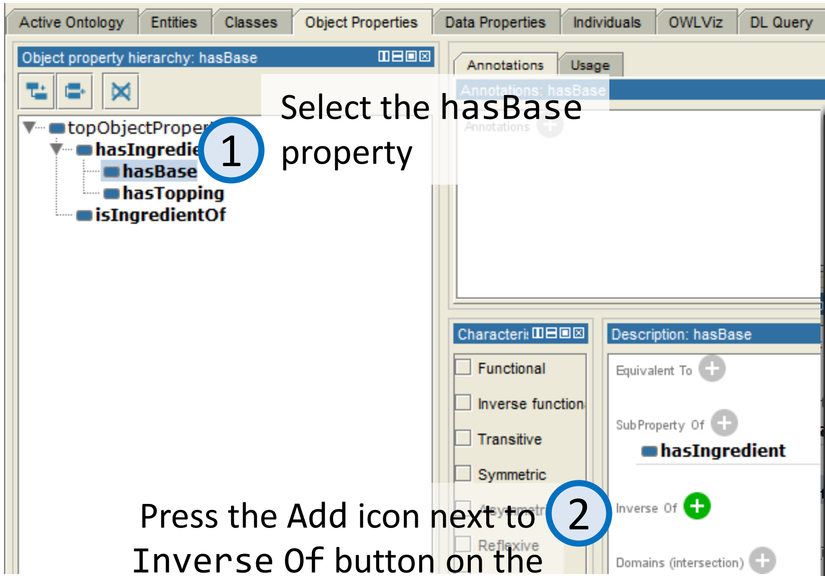
Equivalent To

SubProperty Of

Inverse Of
- isIngredientOf

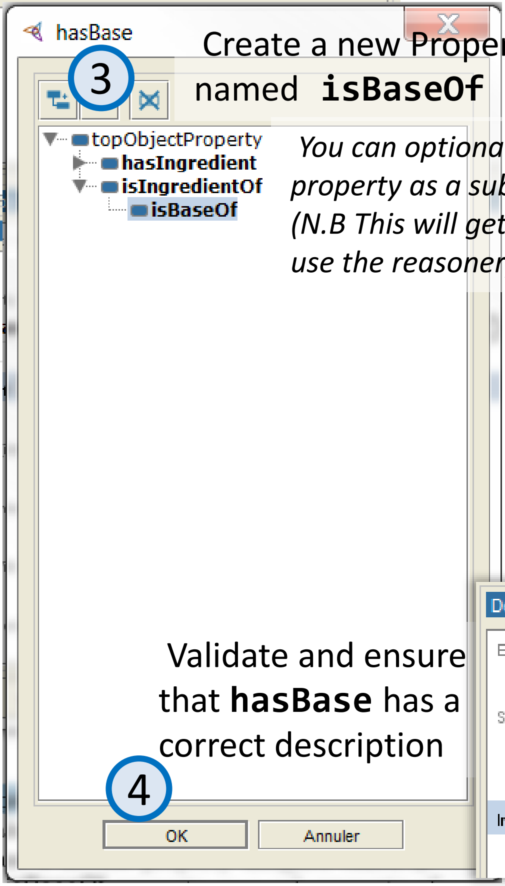
Create inverse properties (continued)

Create an Object Property **isBaseOf** as the inverse of the **hasBase** property



1 Select the hasBase property

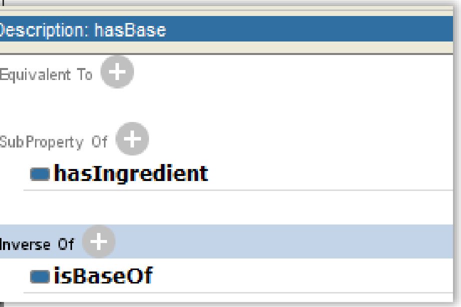
2 Press the Add icon next to Inverse Of button on the Property Description view



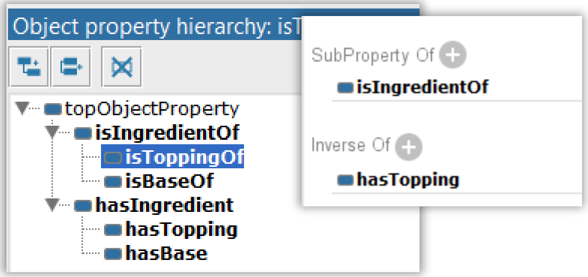
3 Create a new Property named **isBaseOf**

You can optionally place the new isBaseOf property as a sub-property of isIngredientOf (N.B This will get inferred later anyway when you use the reasoner).

4 Validate and ensure that **hasBase** has a correct description

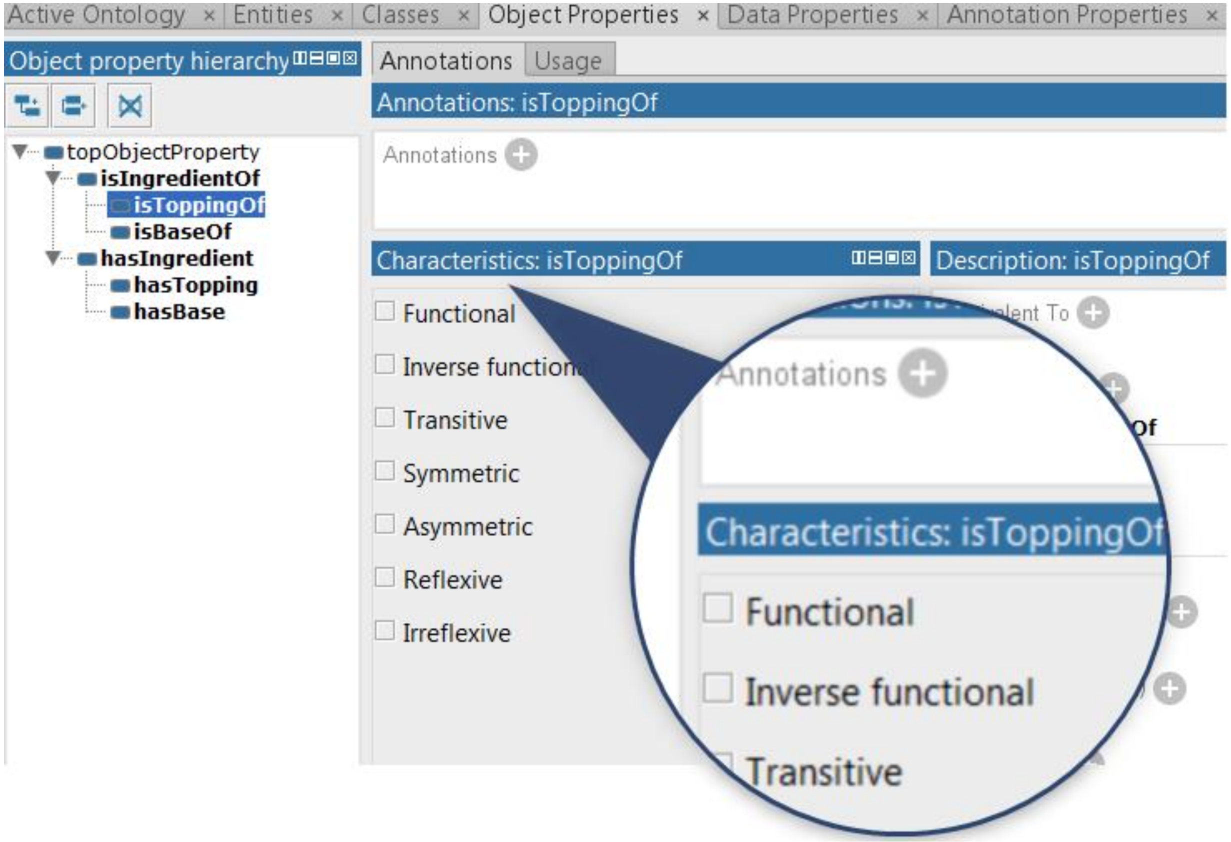


5 Create an Object Property **isToppingOf** as the inverse of the **hasTopping** property



Owl Object Property characteristics

- OWL allows the meaning of properties to be enriched through the use of ***property characteristics***.



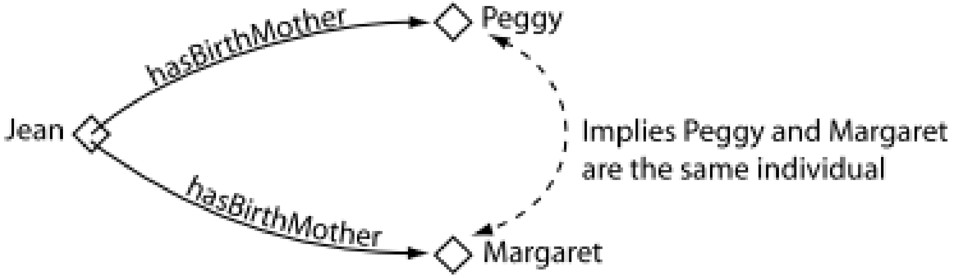
Characteristics: isToppingOf

□ Functional
□ Inverse functional
□ Transitive
□ Symmetric
□ Asymmetric
□ Reflexive
□ Irreflexive

Owl Object Property characteristics

- OWL allows the meaning of properties to be enriched through the use of ***property characteristics***.

- **Functional Properties**
  - If a property is functional, for a given individual, there can be at most one individual that is related to the individual via the property.
  - Example : **`hasBirthMother`** a functional property : something can only have **one** birth mother



if **`Peggy`** and **`Margaret`** were explicitly stated to be two different individuals then the above statements would lead to an inconsistency.

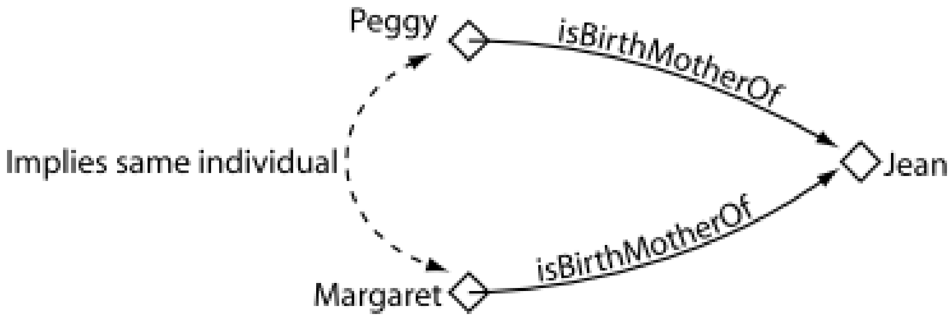
Owl Object Property characteristics

- **Inverse Functional Properties**
  - If a property is inverse functional then it means that the inverse property is functional. For a given individual, there can be at most one individual related to that individual via the property.

  - Example :

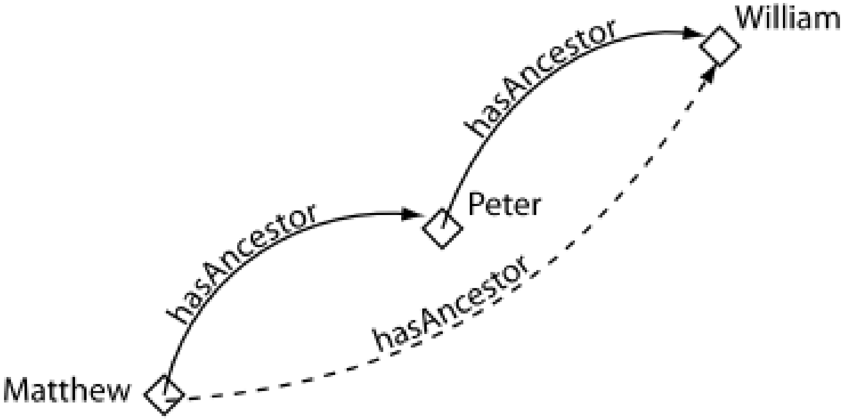
`isBirthMotherOf` : the inverse property of **`hasBirthMother`**

(since **`hasBirthMother`** is functional, **`isBirthMotherOf`** is inverse functional)



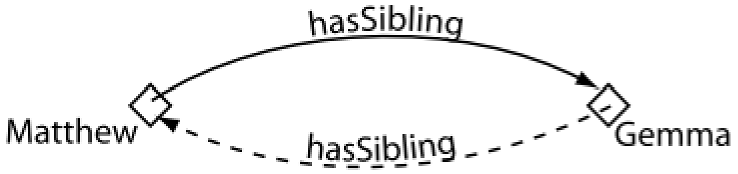
Owl Object Property characteristics

- **Transitive Properties**
  - If a property ***P*** is transitive, and the property relates individual ***a*** to individual ***b***, and also individual ***b*** to individual ***c***, then we can infer that individual ***a*** is related to individual ***c*** via property ***P***.
  - Example : **`hasAncestor`**

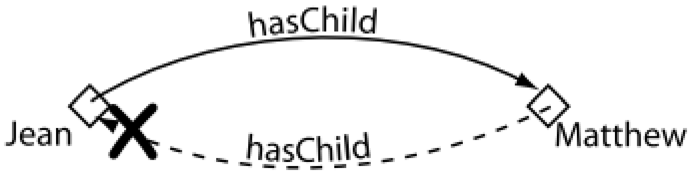


Owl Object Property characteristics

- **Symetric Properties**
  - If a property ***P*** is symmetric, and the property relates individual ***a*** to individual ***b*** then individual ***b*** **is also** related to individual ***a*** via property ***P***.
  - Example : **`hasSibling`**

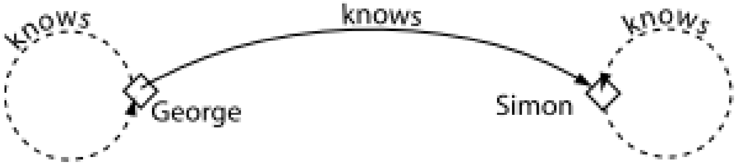


- **Asymetric Properties**
  - If a property ***P*** is asymmetric, and the property relates individual ***a*** to individual ***b*** then individual ***b*** **cannot** be related to individual ***a*** via property ***P***.
  - Example : **`hasChild`**

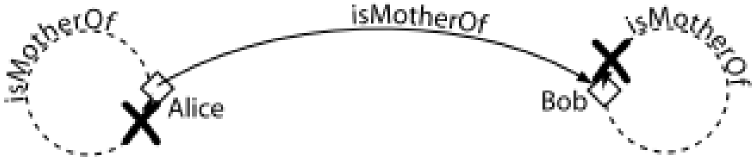


Owl Object Property characteristics

- **Reflexive Properties**
  - A property ***P*** is said to be reflexive when the property must relate individual ***a*** to itself.
  - Example : **knows**

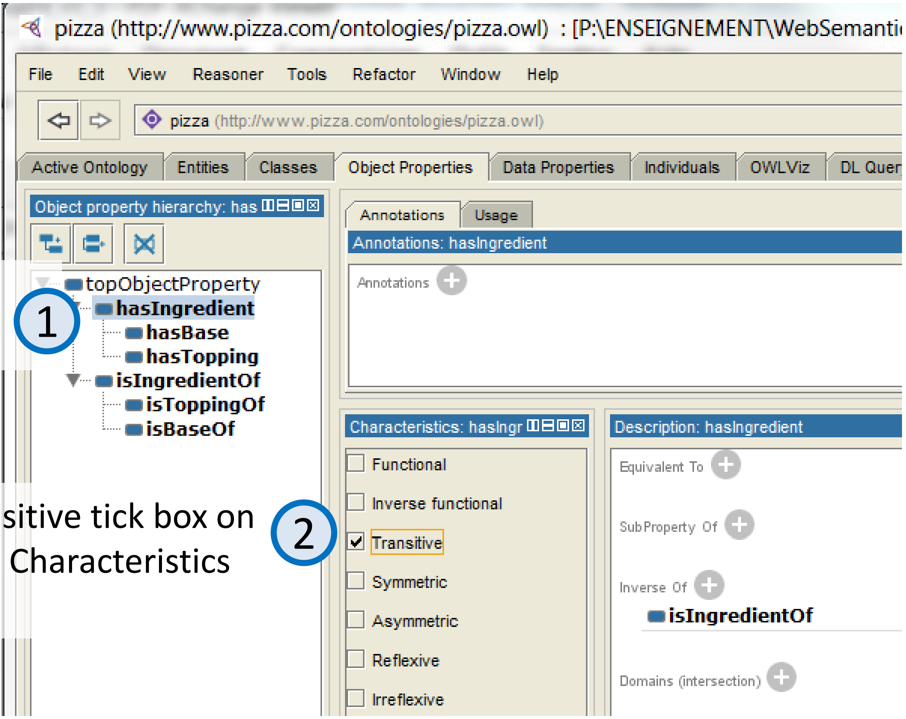


- **Irreflexive Properties**
  - If a property ***P*** is irreflexive, it can be described as a property that relates an individual ***a*** to individual ***b***, where individual ***a*** and individual ***b*** are not the same.
  - Example : **isMotherOf**



Changing property characteristics

Make the **hasIngredient** property transitive



- topObjectProperty
  - hasIngredient
    - hasBase
    - hasTopping
  - isIngredientOf
    - isToppingOf
    - isBaseOf

□ Functional
□ Inverse functional
☑ Transitive
□ Symmetric
□ Asymmetric
□ Reflexive
□ Irreflexive

Inverse Of: isIngredientOf

1 Select the **hasIngredient** property

2 Tick the Transitive tick box on the Property Characteristics View

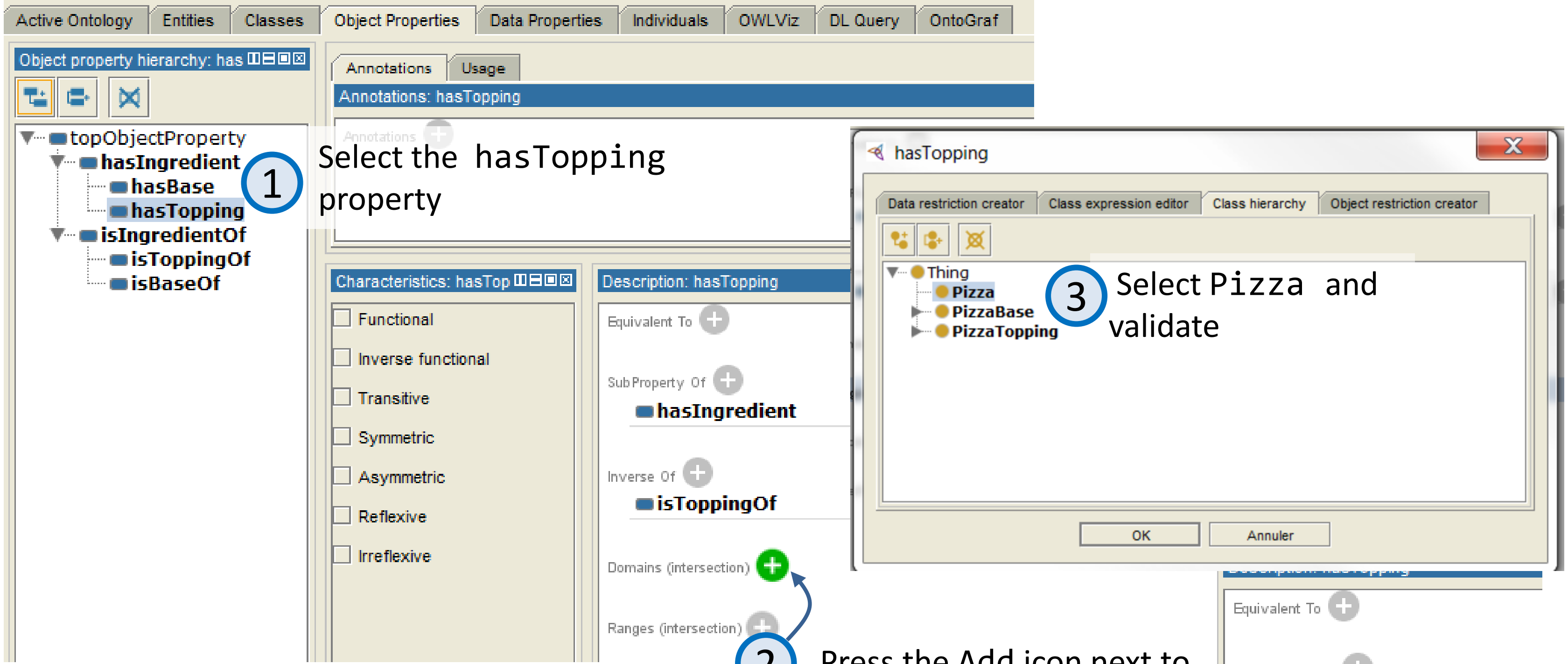
If a property is transitive then its inverse property should also be transitive.

3 Select the `isIngredientOf` property, which is the inverse of `hasIngredient`. Ensure that the transitive tick box is ticked. *this must be done manually in Protégé 4+. However, the reasoner will assume that if a property is transitive, its inverse property is also a transitive.*

4 Make the **hasBase** property functional

Specify Domain and Range

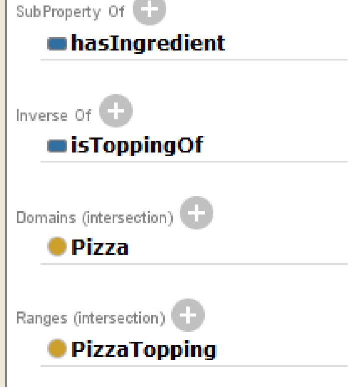
Specify the **Pizza** class as being the domain of the **hasTopping** property



1 Select the hasTopping property

3 Select Pizza and validate

2 Press the Add icon next to Domain button on the Property Description view

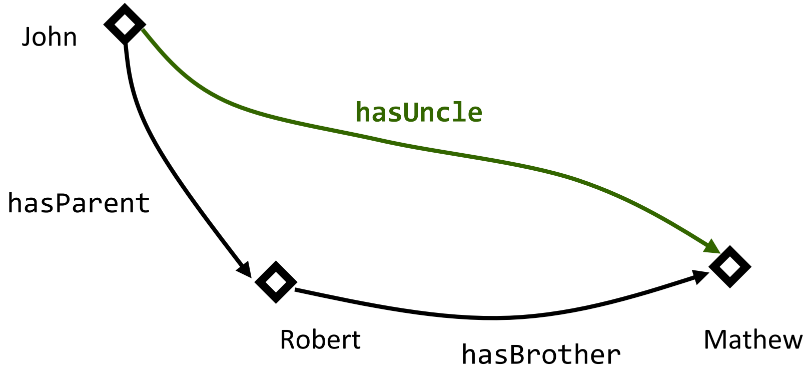


4 Specify the **PizzaTopping** class as being the range of the **hasTopping** property

5 Ensure the **hasTopping** description is correct

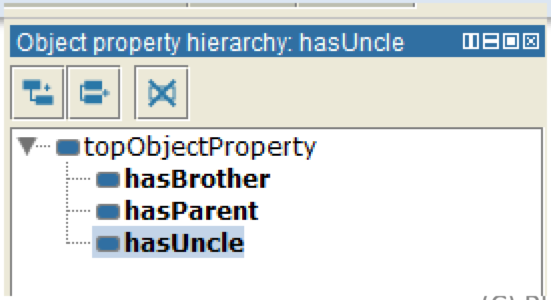
OWL2 Property chains

- a way to define a property in terms of a chain of object properties that connect resources.



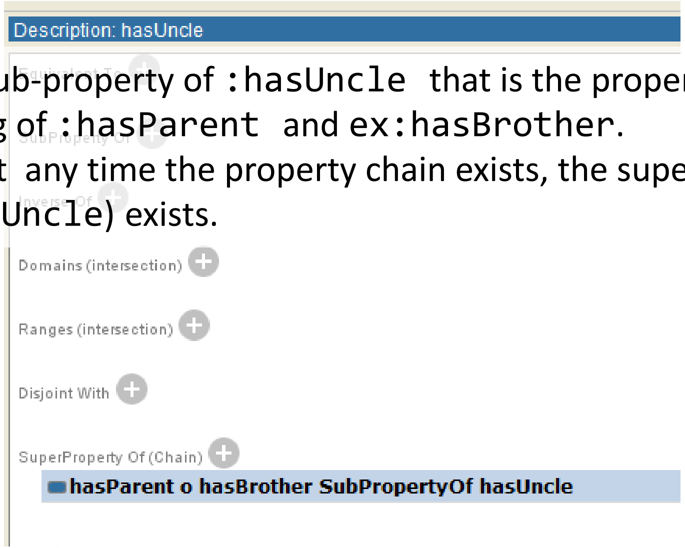
```
:John rdf:type ex:Person;
          :hasParent ex:Robert.
:Robert rdf:type :Person;
        :hasBrother :Mathew.
:Mathew rdf:type ex:Person.

:hasUncle rdf:type owl:ObjectProperty.
:hasParent rdf:type owl:ObjectProperty.
:hasBrother rdf:type owl:ObjectProperty.

[rdfs:subPropertyOf ex:hasUncle;
   owl:propertyChain (
     :hasParent
     :hasBrother
   )
].
```

there exists a sub-property of `:hasUncle` that is the property chain consisting of `:hasParent` and `ex:hasBrother`. This means that any time the property chain exists, the super-property (`:hasUncle`) exists.



Object property hierarchy: hasUncle

- topObjectProperty
  - hasBrother
  - hasParent
  - hasUncle



Description: hasUncle

Domains (intersection)

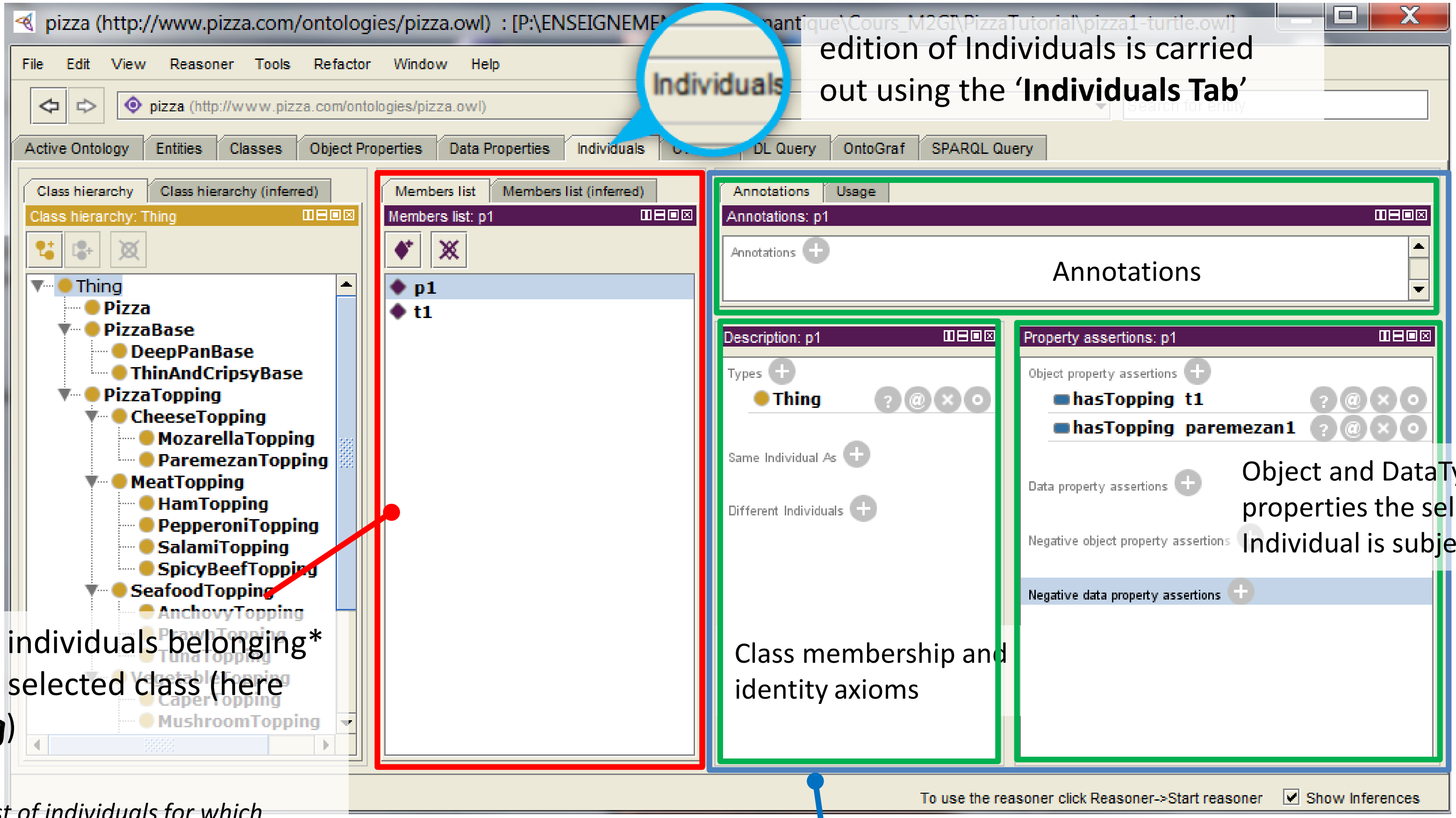
Ranges (intersection)

Disjoint With

SuperProperty Of (Chain)

hasParent o hasBrother SubPropertyOf hasUncle

Individuals Tab



edition of Individuals is carried out using the '**Individuals Tab**'

Annotations

Object and DataType properties the selected Individual is subject of.

Class membership and identity axioms

List of individuals belonging* to the selected class (here **Thing**)

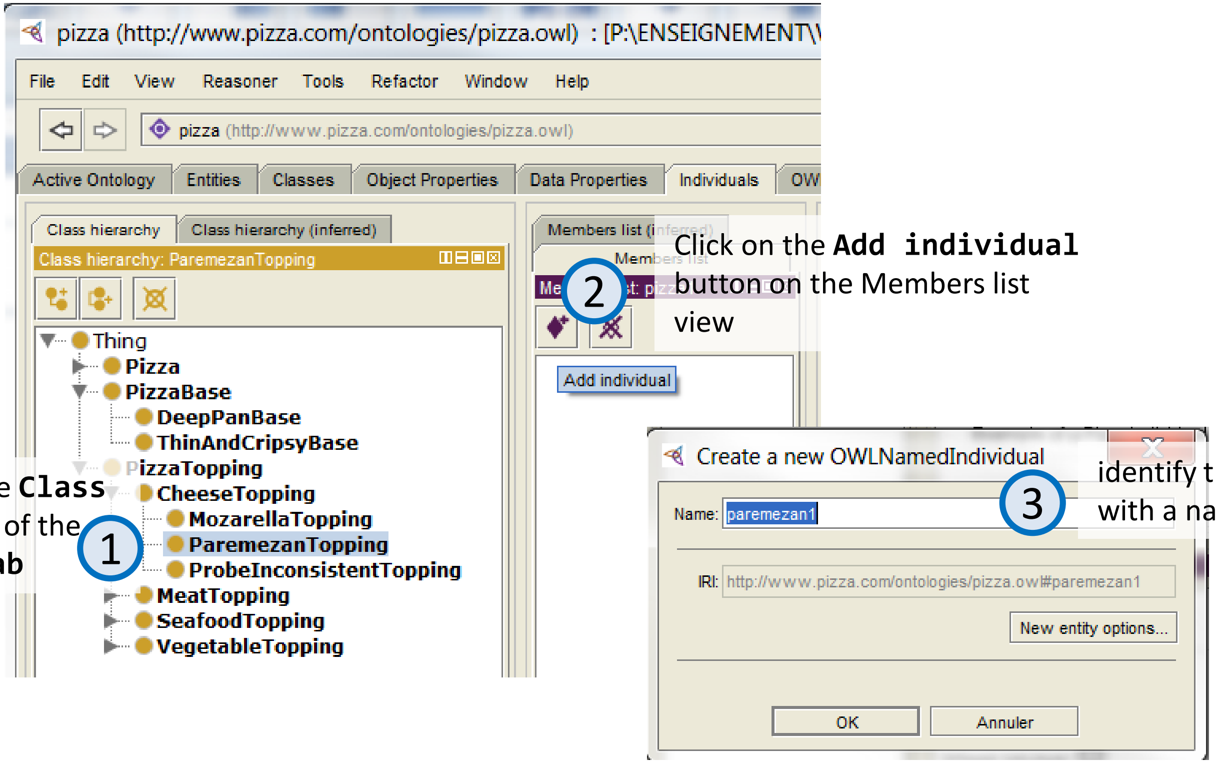
* *the list of individuals for which membership is asserted*

Selected Individual (**p1**) description:
OWL axioms the selected Individual is subject of.

see the video to configure Individual View.

Creating new Individuals

Create a new individual **paremezan1** in the class **ParemezanTopping**



pizza (http://www.pizza.com/ontologies/pizza.owl) : [P:\ENSEIGNEMENT\

Create a new OWLNamedIndividual

Name: paremezan1

IRI: http://www.pizza.com/ontologies/pizza.owl#paremezan1

1 Select a class in the **Class hierarchy** view of the **Individuals Tab**

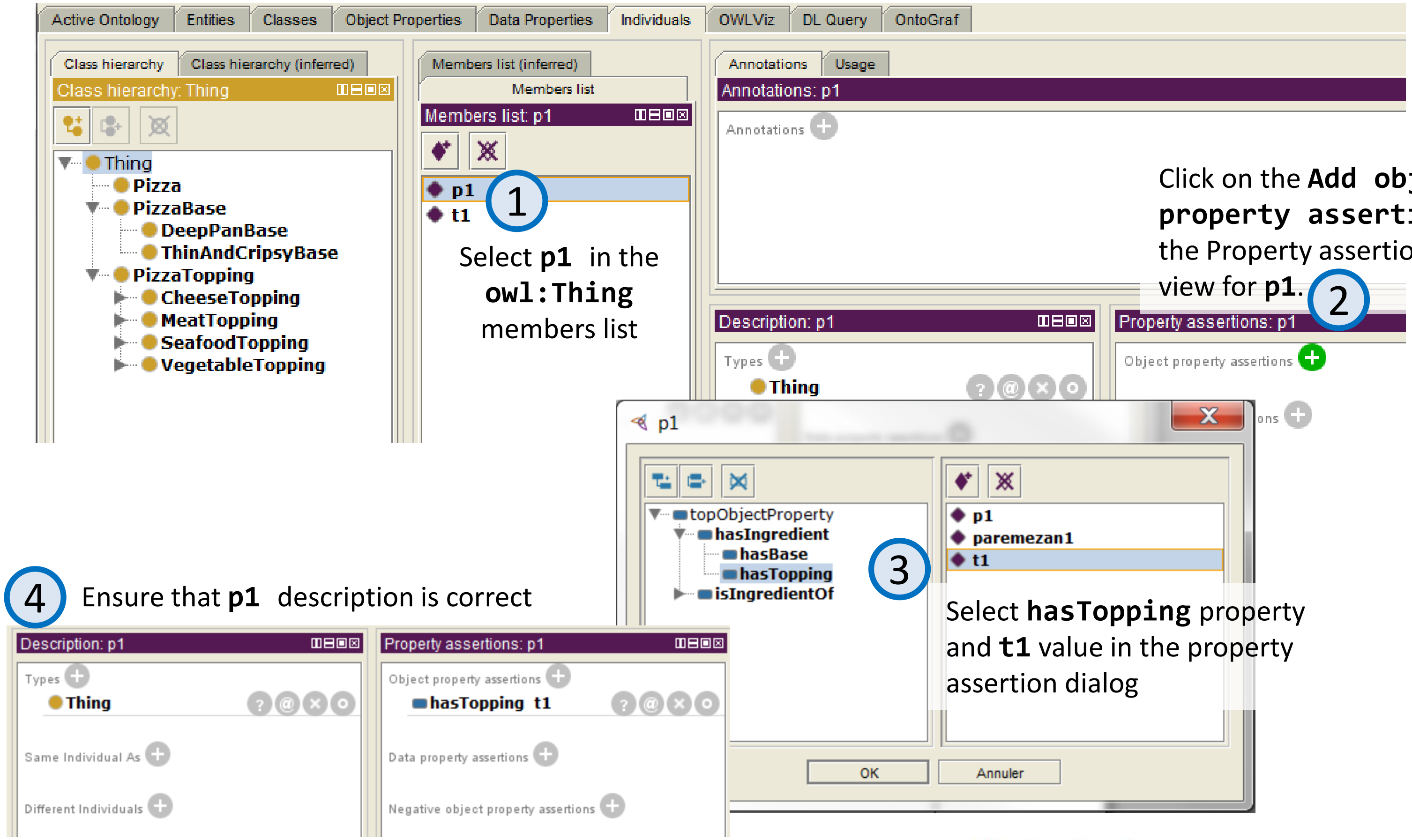
2 Click on the **Add individual** button on the Members list view

3 identify the new individual with a name

4 Create new individuals **p1**, **t1** in the class **owl:Thing**

Creating new Individuals

Create a new **hasTopping** relation in between individual **p1** and individual **t1**



1 Select **p1** in the **owl:Thing** members list

2 Click on the **Add object property assertion** in the Property assertions view for **p1**.

3 Select **hasTopping** property and **t1** value in the property assertion dialog

4 Ensure that **p1** description is correct

5 Let's do some (basic) semantic reasoning

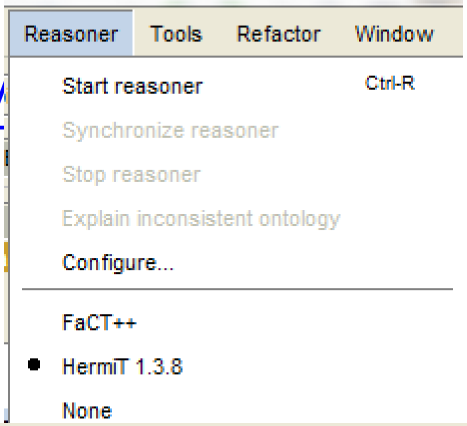
OWL Reasoners

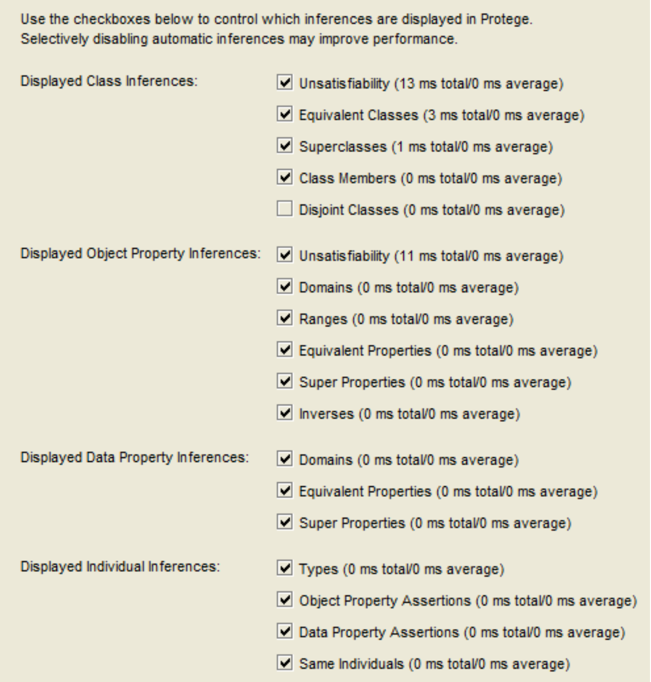
- ontologies that are described using OWL-DL can be processed by a ***reasoner***.
  - thanks to the semantics of the description language the reasoner can deduce new facts from the facts asserted in the ontology.
  - example of services offered by a reasoner
    - **classification**
      - test whether or not one class is a subclass of another class.
        → to compute the inferred ontology class hierarchy
    - **consistency checking**
      - Based on the description (conditions) of a class the reasoner can check whether or not it is possible for the class to have any instances.
        → class is *inconsistent* if it cannot possibly have any instances
    - **realization**
      - find the classes of individuals

Reasoners in Protege

- two reasoners integred to Protégé 4.3
  - FaCT++ http://owl.man.ac.uk/factplusplus/
    - C++ reasoner
  - Hermit http://hermit-reasoner.com/
    - Java reasoner (OWL-API) based on “hypertableau” calculus

      Boris Motik, Rob Shearer, and Ian Horrocks.
      Hypertableau Reasoning for Description Logics.
      Journal of Artificial Intelligence Research, 36:165-228, 2009.
      http://www.hermit-reasoner.com/publications/msh09hypertableau.pdf





Use the checkboxes below to control which inferences are displayed in Protege.
Selectively disabling automatic inferences may improve performance.

Displayed Class Inferences:
- ☑ Unsatisfiability (13 ms total/0 ms average)
- ☑ Equivalent Classes (3 ms total/0 ms average)
- ☑ Superclasses (1 ms total/0 ms average)
- ☑ Class Members (0 ms total/0 ms average)
- □ Disjoint Classes (0 ms total/0 ms average)

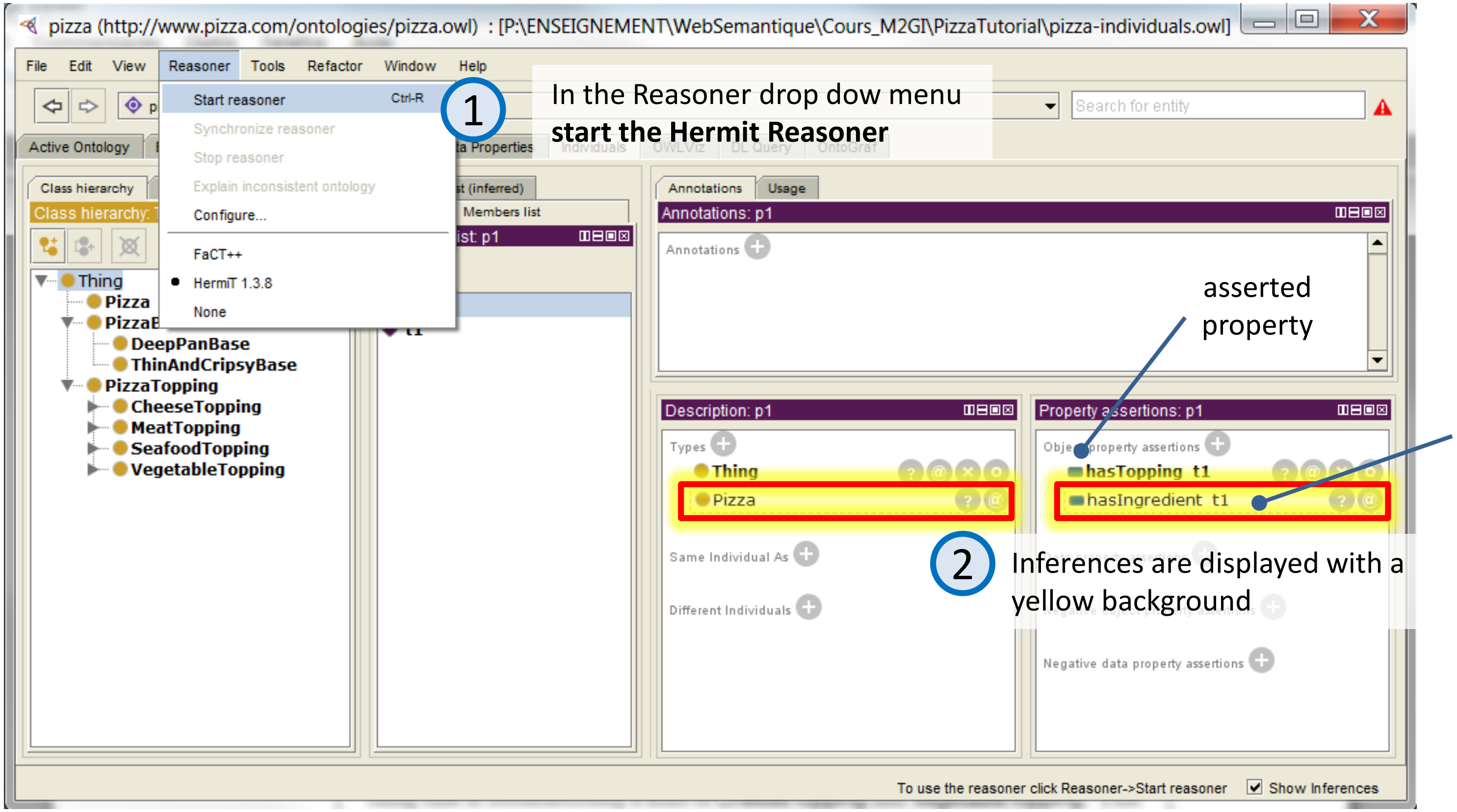
Displayed Object Property Inferences:
- ☑ Unsatisfiability (11 ms total/0 ms average)
- ☑ Domains (0 ms total/0 ms average)
- ☑ Ranges (0 ms total/0 ms average)
- ☑ Equivalent Properties (0 ms total/0 ms average)
- ☑ Super Properties (0 ms total/0 ms average)
- ☑ Inverses (0 ms total/0 ms average)

Displayed Data Property Inferences:
- ☑ Domains (0 ms total/0 ms average)
- ☑ Equivalent Properties (0 ms total/0 ms average)
- ☑ Super Properties (0 ms total/0 ms average)

Displayed Individual Inferences:
- ☑ Types (0 ms total/0 ms average)
- ☑ Object Property Assertions (0 ms total/0 ms average)
- ☑ Data Property Assertions (0 ms total/0 ms average)
- ☑ Same Individuals (0 ms total/0 ms average)

- other reasoners (commercial)
  - Pelet
  - RACER

Reasoning on individuals

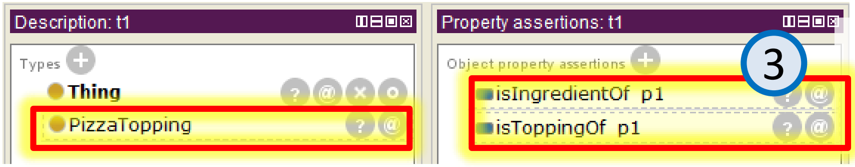


1 In the Reasoner drop dow menu **start the Hermit Reasoner**

asserted property

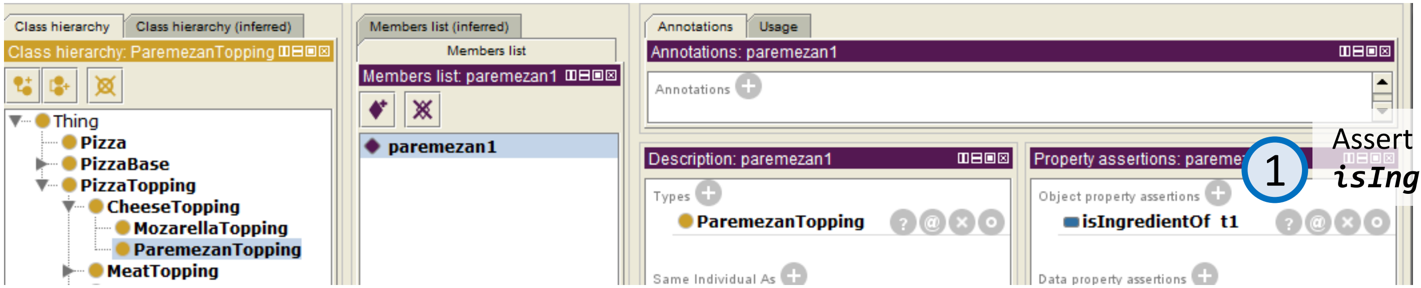
inferred property

2 Inferences are displayed with a yellow background

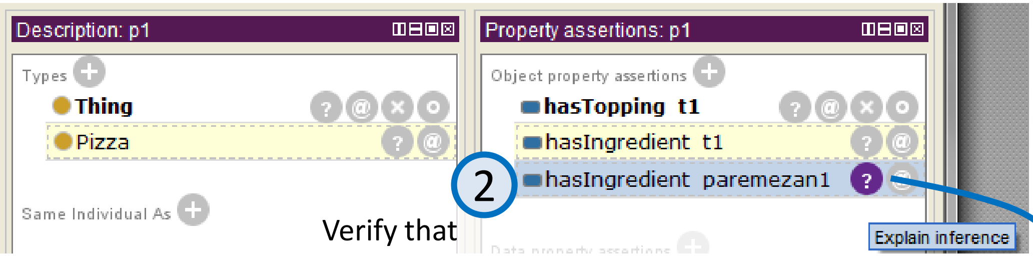


3 Ensure that **t1** as been infered to be a `PizzaTopping` member, an ingredient and atopping of p1.

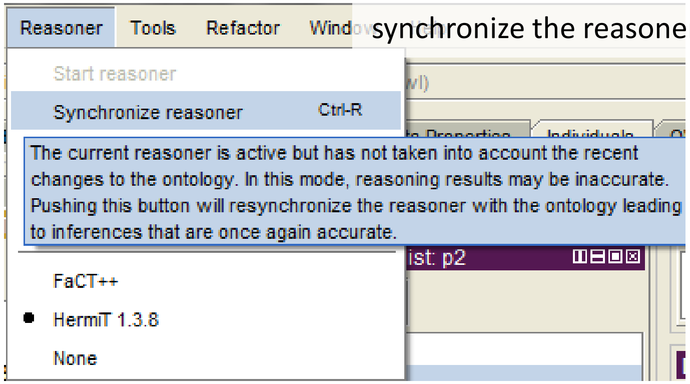
Reasoning on individuals



1 Assert that individual **paremezan1** ***isIngredientOf*** **t1**

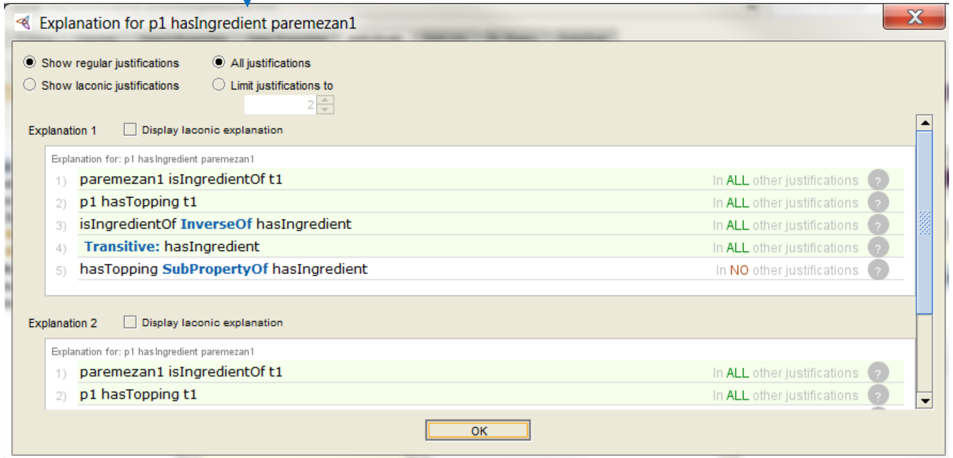


2 Verify that **p1** ***hasIngredient*** **paremezan1** has been inferred. If necessary synchronize the reasoner.



The current reasoner is active but has not taken into account the recent changes to the ontology. In this mode, reasoning results may be inaccurate. Pushing this button will resynchronize the reasoner with the ontology leading to inferences that are once again accurate.

3 Look for explanations about this inference


Explanation for p1 hasIngredient paremezan1

● Show regular justifications ● All justifications
○ Show laconic justifications ○ Limit justifications to 2

Explanation 1 □ Display laconic explanation

Explanation for: p1 hasIngredient paremezan1

1) paremezan1 isIngredientOf t1 — In ALL other justifications
2) p1 hasTopping t1 — In ALL other justifications
3) isIngredientOf **InverseOf** hasIngredient — In ALL other justifications
4) **Transitive:** hasIngredient — In ALL other justifications
5) hasTopping **SubPropertyOf** hasIngredient — In NO other justifications

Explanation 2 □ Display laconic explanation

Explanation for: p1 hasIngredient paremezan1

1) paremezan1 isIngredientOf t1 — In ALL other justifications
2) p1 hasTopping t1 — In ALL other justifications

OK

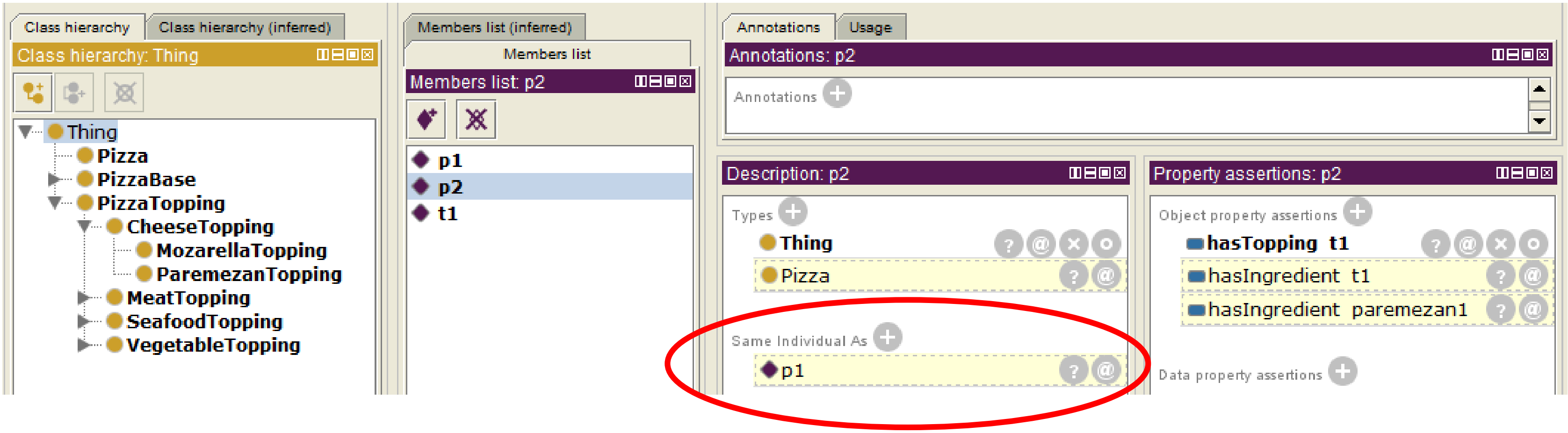
Reasoning on individuals

1. Create a new individual **p2** in the class **`owl:Thing`**

2. Assert that **`p2 hasTopping t1`**

3. Make **`hasTopping`** inverseFonctional

4. Run the reasoner and verify that **p2** is the same as **p1**



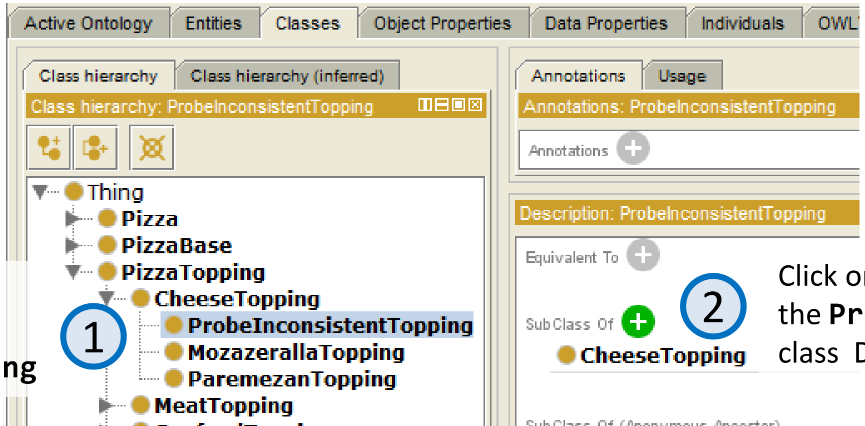
Types: Thing, Pizza

Same Individual As: p1

Object property assertions:
hasTopping t1
hasIngredient t1
hasIngredient paremezan1

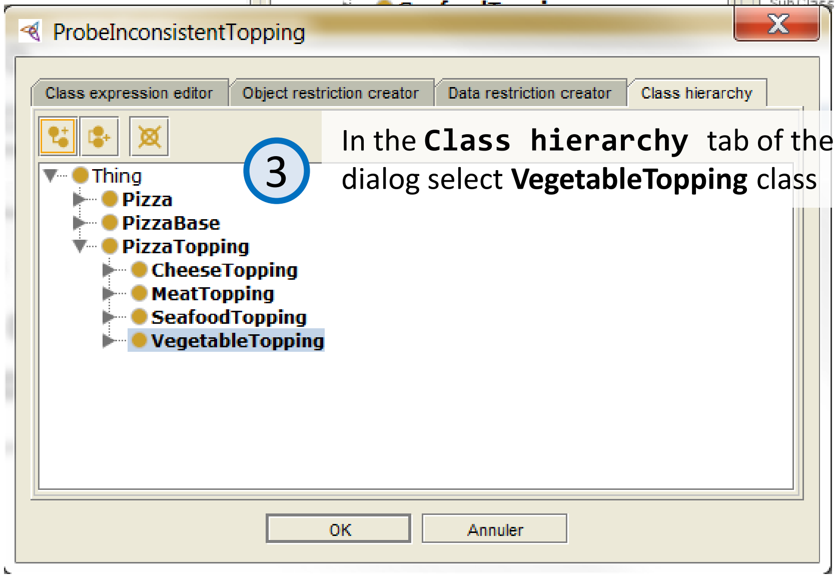
Testing for Inconsistent Classes

To demonstrate the use of the reasoner in detecting inconsistencies in the ontology create a **ProbeInconsistentTopping** class that is a subclass of both **CheeseTopping** and also **VegetableTopping**.



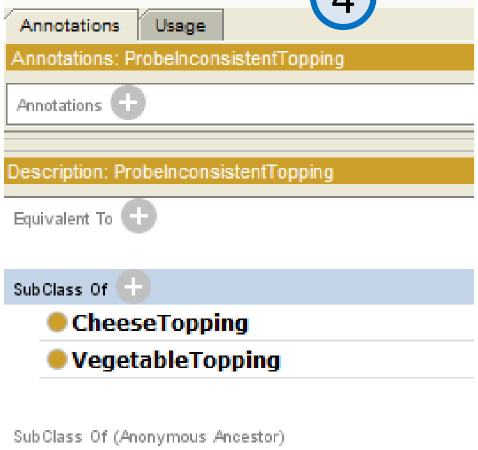
1 Create a subclass of **CheeseTopping** named **ProbeInconsistentTopping**

2 Click on the **Add SubClass of** button on the **ProbeInconsistentTopping** class Description View.

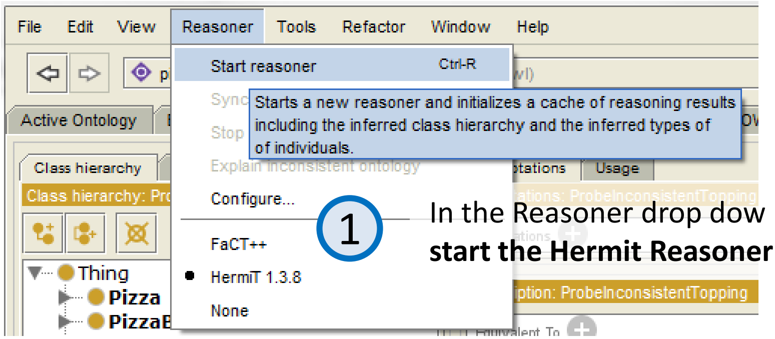


3 In the **Class hierarchy** tab of the dialog select **VegetableTopping** class

4 ensure that the **ProbeInconsistentTopping** class description is correct.

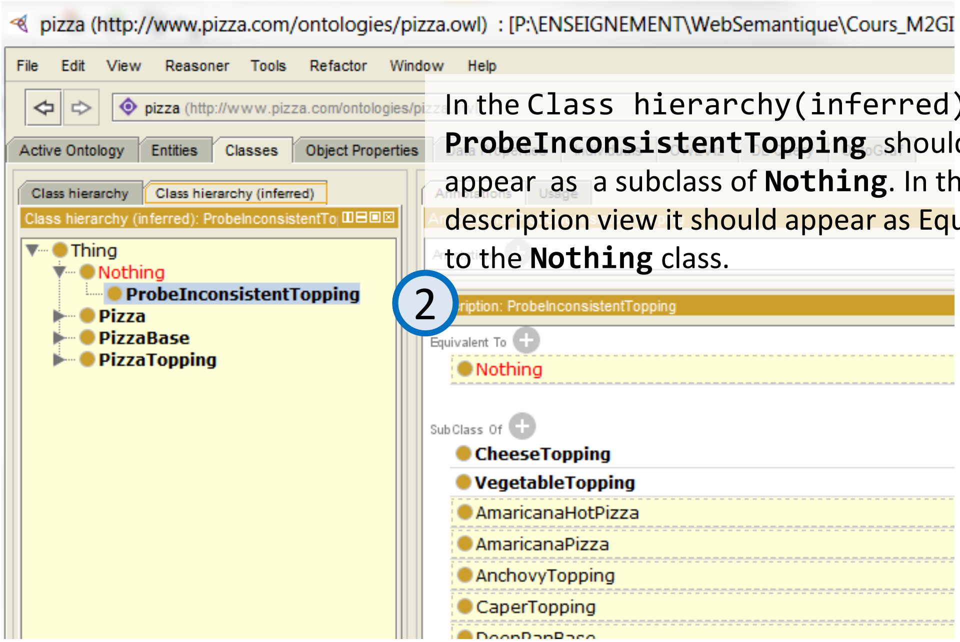

Annotations
Usage
Annotations: ProbeInconsistentTopping
Annotations
Description: ProbeInconsistentTopping
Equivalent To
SubClass Of
CheeseTopping
VegetableTopping
SubClass Of (Anonymous Ancestor)

Testing for Inconsistent Classes



Starts a new reasoner and initializes a cache of reasoning results including the inferred class hierarchy and the inferred types of individuals.

1 In the Reasoner drop dow menu **start the Hermit Reasoner**


pizza (http://www.pizza.com/ontologies/pizza.owl) : [P:\ENSEIGNEMENT\WebSemantique\Cours_M2GI

In the `Class hierarchy(inferred)` tab, **`ProbeInconsistentTopping`** should appear as a subclass of **`Nothing`**. In the description view it should appear as Equivalent to the **`Nothing`** class.

2

`Nothing ???`

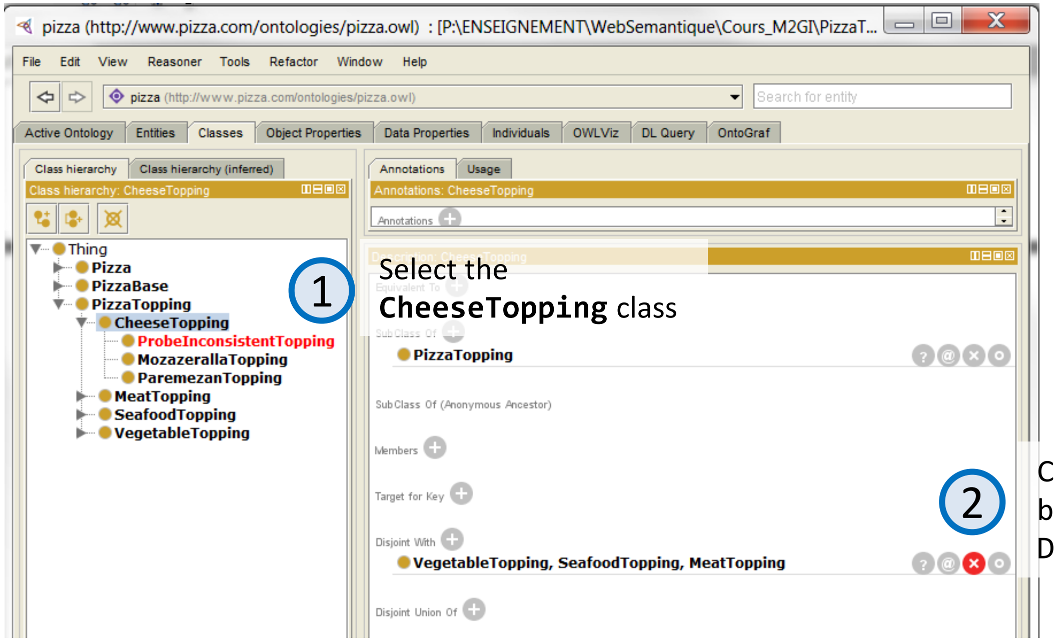
`owl:Nothing` is a predefined class whose extension is the empty set. Consequently, **`owl:Nothing`** is a subclass of every class and a class equivalent to **`owl:Nothing`** is inconsistent, it can't have any instances.

Why **`ProbeInconsistentTopping`** has been found as inconsistent ?

because its superclasses `VegetableTopping` and `CheeseTopping` are disjoint from each other →individuals that are members of the class `CheeseTopping` cannot be members of the class `VegetableTopping` and vice-versa.

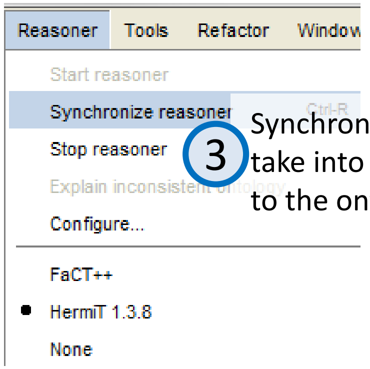
Testing for Inconsistent Classes

Remove the disjoint statement between **CheeseTopping** and **VegetableTopping** to see what happens.


pizza (http://www.pizza.com/ontologies/pizza.owl) : [P:\ENSEIGNEMENT\WebSemantique\Cours_M2GI\PizzaT...
File Edit View Reasoner Tools Refactor Window Help
pizza (http://www.pizza.com/ontologies/pizza.owl)
Search for entity
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Class hierarchy: CheeseTopping
Thing
Pizza
PizzaBase
PizzaTopping
CheeseTopping
ProbeInconsistentTopping
MozazerallaTopping
ParemezanTopping
MeatTopping
SeafoodTopping
VegetableTopping
Annotations Usage
Annotations: CheeseTopping
Annotations
Equivalent To
SubClass Of
PizzaTopping
SubClass Of (Anonymous Ancestor)
Members
Target for Key
Disjoint With
VegetableTopping, SeafoodTopping, MeatTopping
Disjoint Union Of

1 Select the **CheeseTopping** class

2 Click on the **Remove Disjoint With** button on the **CheeseTopping** class `Description View`.


Reasoner Tools Refactor Window
Start reasoner
Synchronize reasoner Ctrl-R
Stop reasoner
Explain inconsistent ontology
Configure...
FaCT++
● HermiT 1.3.8
None

3 Synchronize the reasoner to take into account the change to the ontology

4 Verify that **ProbeInconsistentTopping** is no longer inconsistent.

5 Fix the ontology by making again **CheeseTopping** and its siblings classes disjoint from each other, and delete **ProbeInconsistentTopping** class

- Using properties to describe classes
  - Properties restriction

Properties Restrictions

- In the previous examples, classes were explicitly defined.

  → **named classes**

- In OWL a class can be described or defined by the relationships that its members (individuals) participate in.

  → **properties restrictions** (another kind of classes)

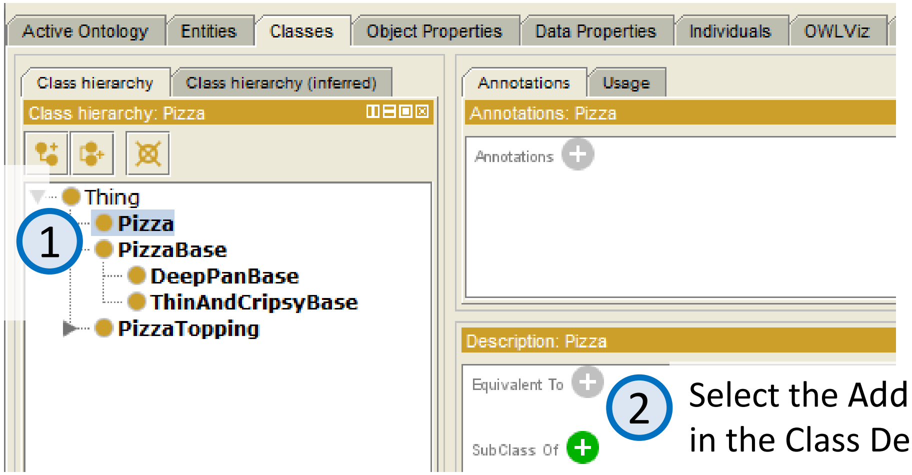
  – examples:
    - The class of individuals that have more than three ***hasTopping*** relationships.
    - The class of individuals that have at least one ***hasTopping*** relationship to individuals that are members of **`MozzarellaTopping`** – i.e. the class of things that have at least one kind of mozzarella topping.
    - The class of individuals that only have ***hasTopping*** relationships to members of **`VegetableTopping`** – i.e. the class of individuals that only have toppings that are vegetable toppings.

Categories of restrictions

- three main categories of **properties restrictions**
  - **Quantifiers Restrictions**
    - Existential Restrictions (**owl:someValuesFrom** restricition ⇔ ∃ quantifier in D.L.)
      - classes of individuals that participate in ***at least one*** relationship along a specified property to individuals that are members of a specified class.
      - ex : *the class of individuals that have at least one (some)* ***hasTopping*** *relationship to members of* ***MozzarellaTopping***
    - Universal Restrictions (**owl:allValuesFrom** restriction ⇔ ∀ quantifier in D.L.)
      - classes of individuals that for a given property ***only*** have relationships along this property to individuals that are members of a specified class.
      - ex: *the class of individuals that only have* ***hasTopping*** *relationships to members of* ***VegetableTopping***.
  - **Cardinality Restrictions**
  - **hasValue Restrictions**

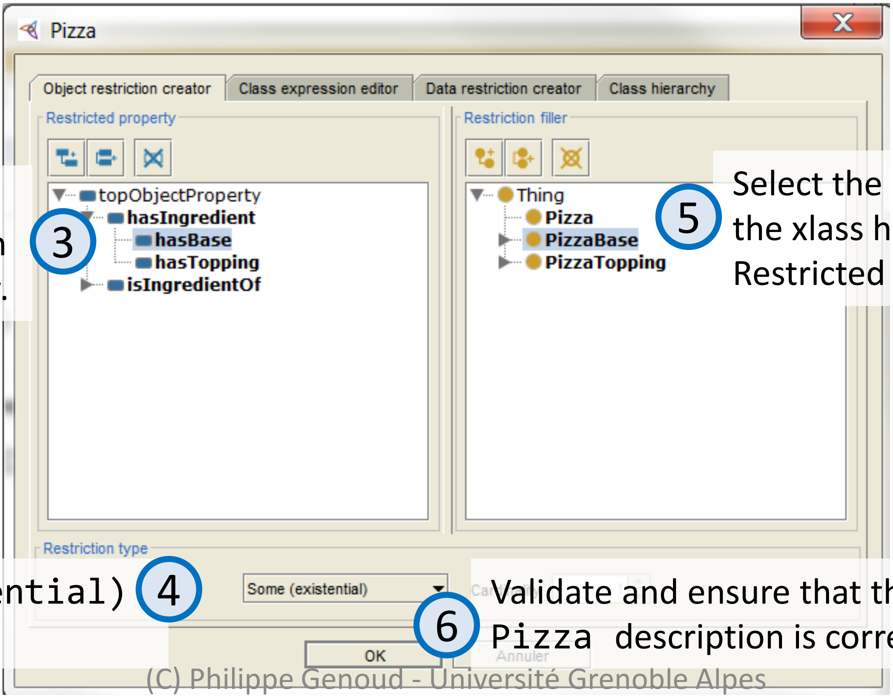
Creating a class with an existential restriction

Add an existential restriction to the **Pizza** class that specifies a **Pizza** must have a **PizzaBase**



1 Select the Pizza class

2 Select the Add icon next to SubClass Of header in the Class Description View .

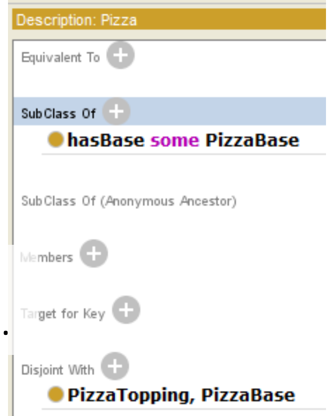


3 Select the hasBase on the property hierarchy in Restricted property view.

4 Select the Some (existential) restriction type.

5 Select the PizzaBase on the xlass hierarchy in Restricted property view.

6 Validate and ensure that the Pizza description is correct.


Description: Pizza

Equivalent To

SubClass Of

hasBase some PizzaBase

SubClass Of (Anonymous Ancestor)

Members

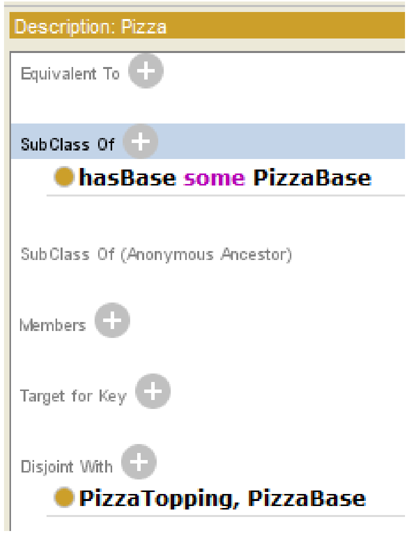
Target for Key

Disjoint With

PizzaTopping, PizzaBase

Interpretation of existential restrictions

Meaning of the restriction



Description: Pizza

Equivalent To +

SubClass Of +

● hasBase some PizzaBase

SubClass Of (Anonymous Ancestor)

Members +

Target for Key +

Disjoint With +

● PizzaTopping, PizzaBase

Restrictions are used in OWL class descriptions to specify *anonymous superclasses* (unnamed classes) of the class being described.

The anonymous class corresponding to a restriction contains all of the individuals that satisfy the restriction – i.e. all of the individuals that have the relationships required to be a member of the class.

Turtle

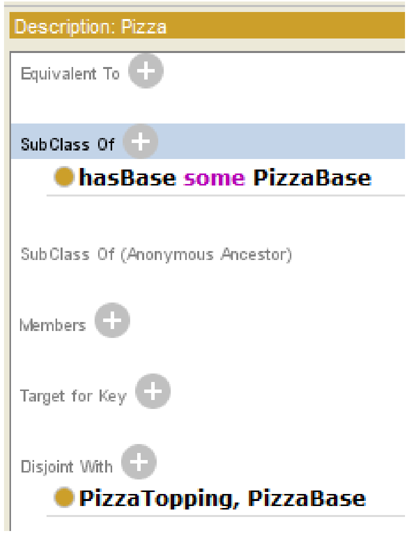
blank node corresponding to an anonymous class

```
:Pizza rdf:type owl:Class ;
       rdfs:subClassOf [
                         rdf:type owl:Restriction ;
                         owl:onProperty :hasBase ;
                         owl:someValuesFrom :PizzaBase
                     ] .
```

RDF/XML

```
<!-- http://www.pizza.com/ontologies/pizza.owl#Pizza -->

<owl:Class rdf:about="http://www.pizza.com/ontologies/pizza.owl#Pizza">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.pizza.com/ontologies/pizza.owl#hasBase"/>
                <owl:someValuesFrom rdf:resource="http://www.pizza.com/ontologies/pizza.owl#PizzaBase"/>
            </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

Interpretation of existential restrictions



Description: Pizza

Equivalent To +

SubClass Of +

● hasBase some PizzaBase

SubClass Of (Anonymous Ancestor)

Members +

Target for Key +

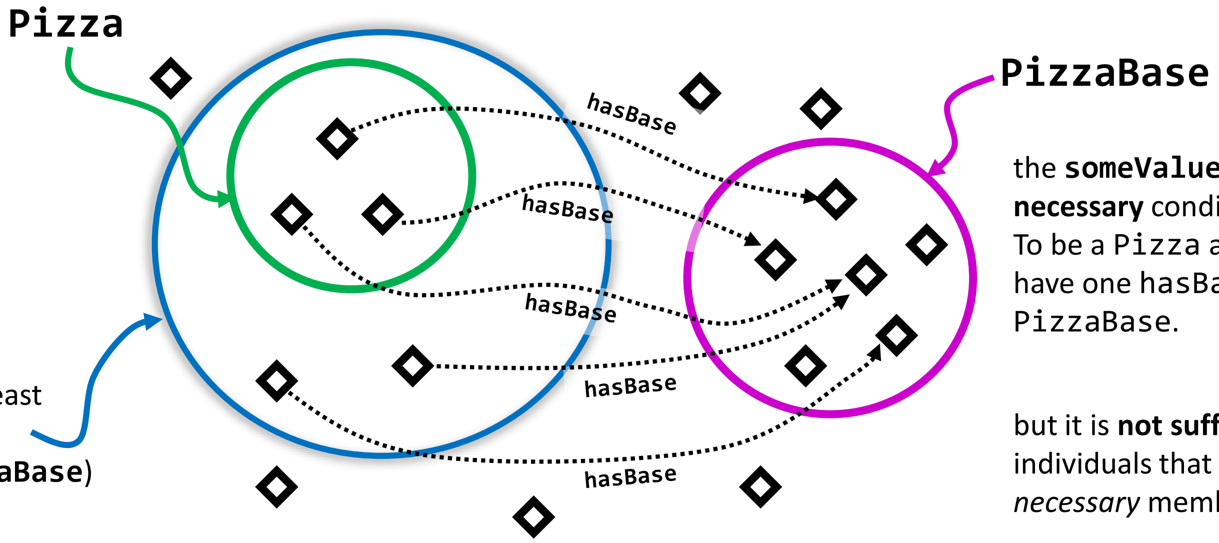
Disjoint With +

● PizzaTopping, PizzaBase

Turtle

```
:Pizza rdf:type owl:Class ;
       rdfs:subClassOf [
                          rdf:type owl:Restriction ;
                          owl:onProperty :hasBase ;
                          owl:someValuesFrom :PizzaBase
                       ] .
```

the class **Pizza** is a subclass of **Thing** and a subclass of the things that have a base which is some kind of **PizzaBase**.



Things that have at least one **PizzaBase** (**hasBase some PizzaBase**)

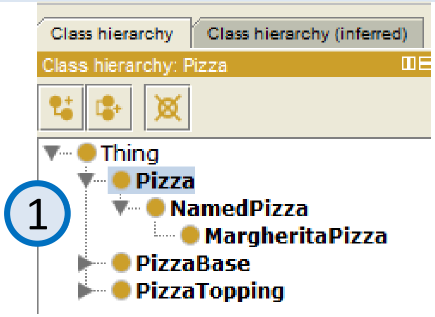
the **someValuesFrom** restriction defines a **necessary** condition :
To be a Pizza an individual ***must*** at least have one hasBase relationship with a PizzaBase.

but it is **not sufficient** :
individuals that have a PizzaBase *are not necessary* members of the Pizza class

Creating subclasses of the Pizza class

Create a subclass of Pizza called NamedPizza, and a subclass of NamedPizza called MargheritaPizza

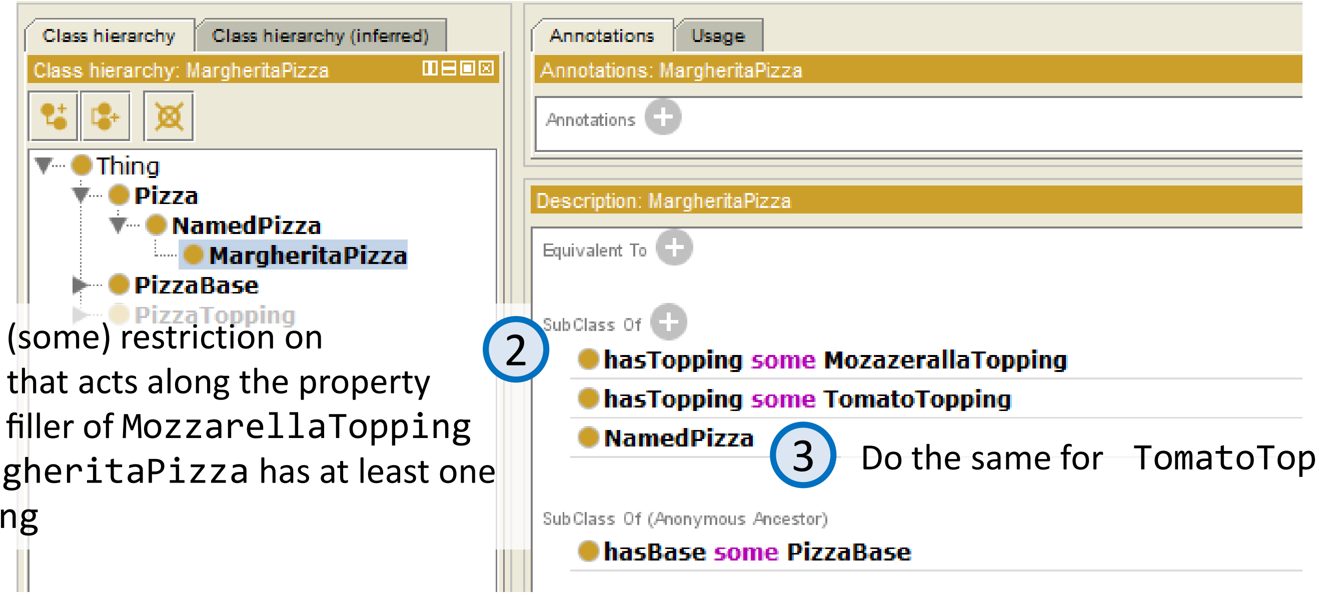
1



Class hierarchy | Class hierarchy (inferred)

Class hierarchy: Pizza

- Thing
  - Pizza
    - NamedPizza
      - MargheritaPizza
  - PizzaBase
  - PizzaTopping



Class hierarchy | Class hierarchy (inferred)

Class hierarchy: MargheritaPizza

- Thing
  - Pizza
    - NamedPizza
      - MargheritaPizza
  - PizzaBase
  - PizzaTopping

Annotations | Usage

Annotations: MargheritaPizza

Annotations

Description: MargheritaPizza

Equivalent To

SubClass Of

- hasTopping some MozazerallaTopping
- hasTopping some TomatoTopping
- NamedPizza

SubClass Of (Anonymous Ancestor)

- hasBase some PizzaBase

2

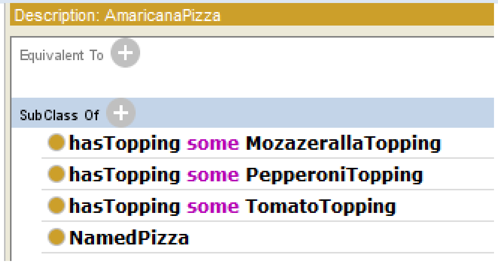
Create an existential (some) restriction on MargheritaPizza that acts along the property hasTopping with a filler of MozzarellaTopping to specify that a MargheritaPizza has at least one MozzarellaTopping

3

Do the same for TomatoTopping

Creating other subclasses of NamedPizza

Now create the class to represent an Americana Pizza, which has toppings of pepperoni, mozzarella and tomato.

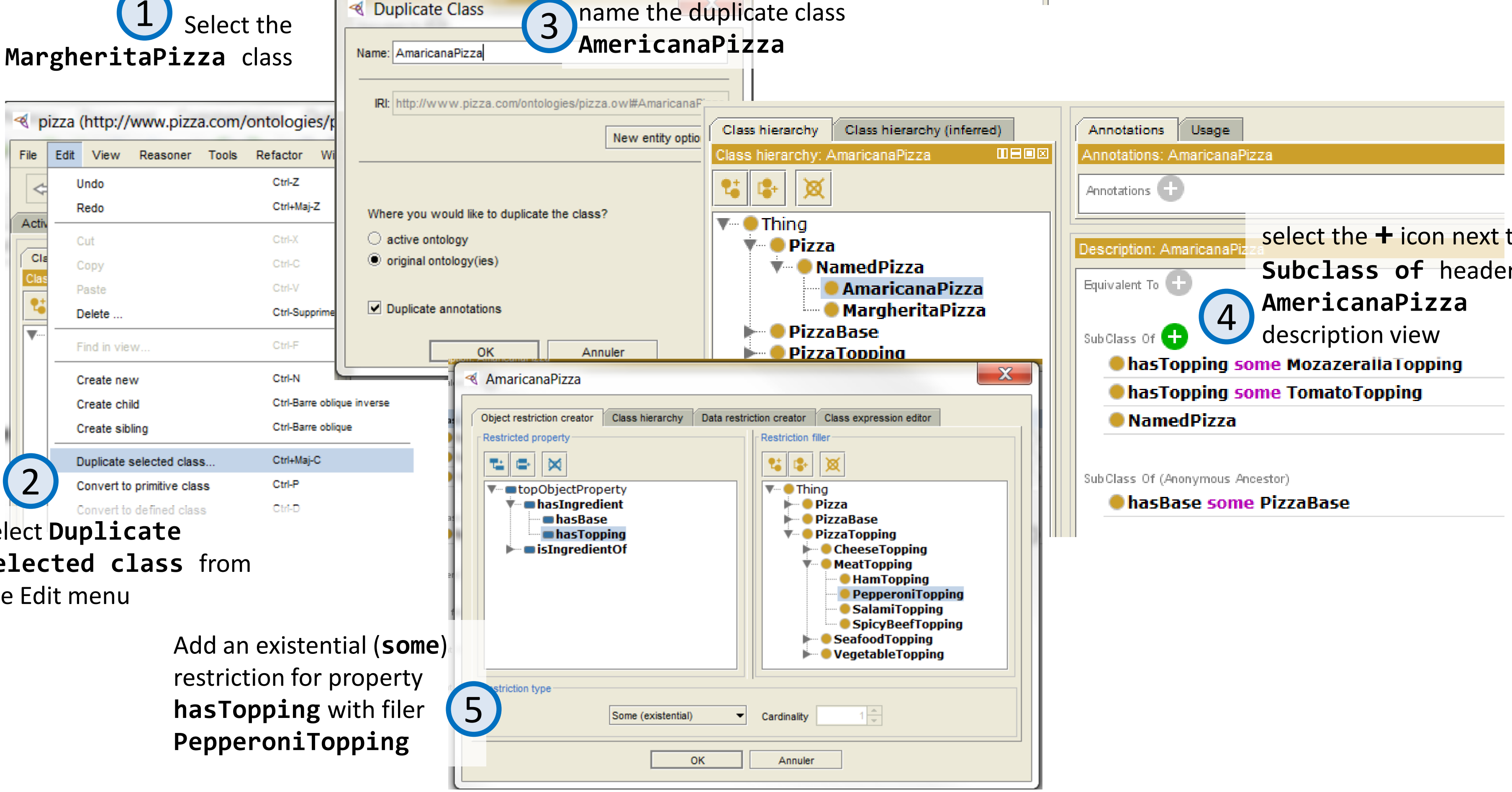


Description: AmaricanaPizza

Equivalent To

SubClass Of

- hasTopping some MozazerallaTopping
- hasTopping some PepperoniTopping
- hasTopping some TomatoTopping
- NamedPizza



1 Select the **MargheritaPizza** class

2 Select **Duplicate selected class** from the Edit menu

3 name the duplicate class **AmericanaPizza**

4 select the **+** icon next to **Subclass of** header in th **AmericanaPizza** description view

5 Add an existential (**some**) restriction for property **hasTopping** with filer **PepperoniTopping**

pizza (http://www.pizza.com/ontologies/p
File
Edit
View
Reasoner
Tools
Refactor
Wi
Undo Ctrl-Z
Redo Ctrl+Maj-Z
Cut Ctrl-X
Copy Ctrl-C
Paste Ctrl-V
Delete ... Ctrl-Supprime
Find in view... Ctrl-F
Create new Ctrl-N
Create child Ctrl-Barre oblique inverse
Create sibling Ctrl-Barre oblique
Duplicate selected class... Ctrl+Maj-C
Convert to primitive class Ctrl-P
Convert to defined class Ctrl-D

Duplicate Class

Name: AmericanaPizza

IRI: http://www.pizza.com/ontologies/pizza.owl#AmericanaP

New entity optio

Where you would like to duplicate the class?

○ active ontology

● original ontology(ies)

☑ Duplicate annotations

OK Annuler

Description: AmaricanaPizza

Equivalent To

SubClass Of

- hasTopping some MozazerallaTopping
- hasTopping some TomatoTopping
- NamedPizza

SubClass Of (Anonymous Ancestor)

- hasBase some PizzaBase

AmericanaPizza

Restriction type: Some (existential) Cardinality 1

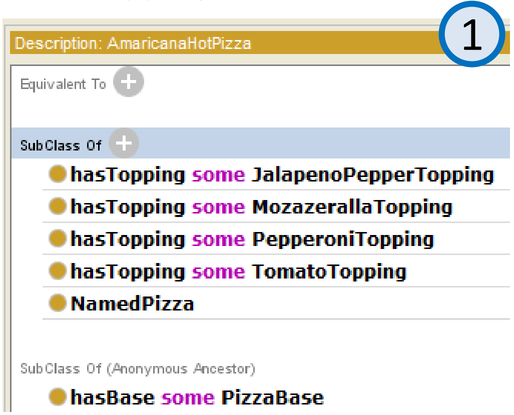
OK Annuler

Creating other subclasses of NamedPizza

Create an **AmericanaHotPizza** class

*same topping as **AmericanaPizza** + Jalapeno pepper*

1

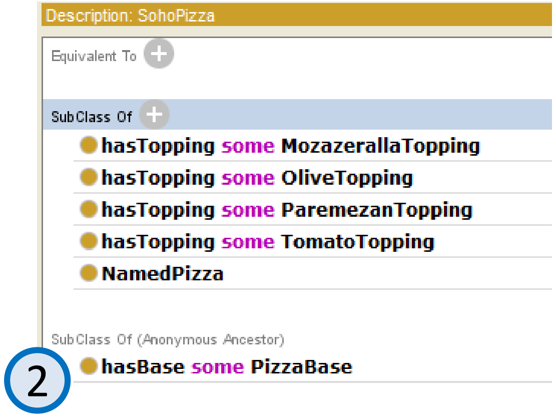


Description: AmaricanaHotPizza

Equivalent To

SubClass Of
- hasTopping some JalapenoPepperTopping
- hasTopping some MozazerallaTopping
- hasTopping some PepperoniTopping
- hasTopping some TomatoTopping
- NamedPizza

SubClass Of (Anonymous Ancestor)
- hasBase some PizzaBase



Description: SohoPizza

Equivalent To

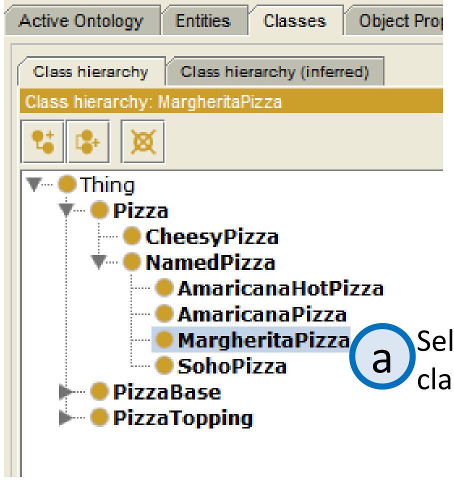
SubClass Of
- hasTopping some MozazerallaTopping
- hasTopping some OliveTopping
- hasTopping some ParemezanTopping
- hasTopping some TomatoTopping
- NamedPizza

SubClass Of (Anonymous Ancestor)
- hasBase some PizzaBase

Create an **SohoPizza** class 2

*same topping as **MagheritaPizzaPizza** + olives+ parmezan cheese*

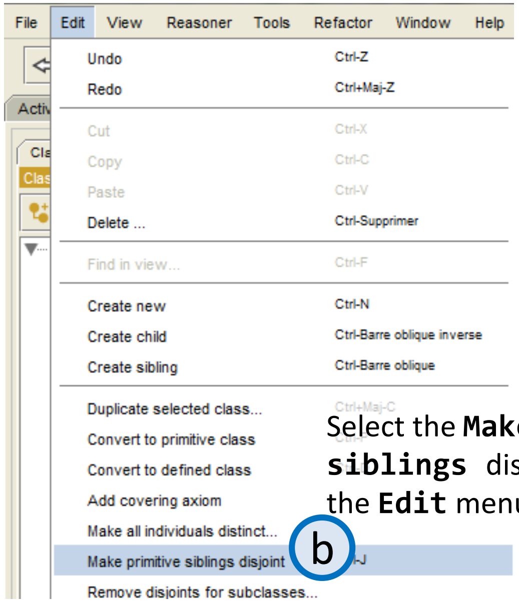
3 Make subclasses of **NamedPizza** disjoint from each other



Class hierarchy: MargheritaPizza

- Thing
  - Pizza
    - CheesyPizza
    - NamedPizza
      - AmaricanaHotPizza
      - AmaricanaPizza
      - MargheritaPizza
      - SohoPizza
  - PizzaBase
  - PizzaTopping

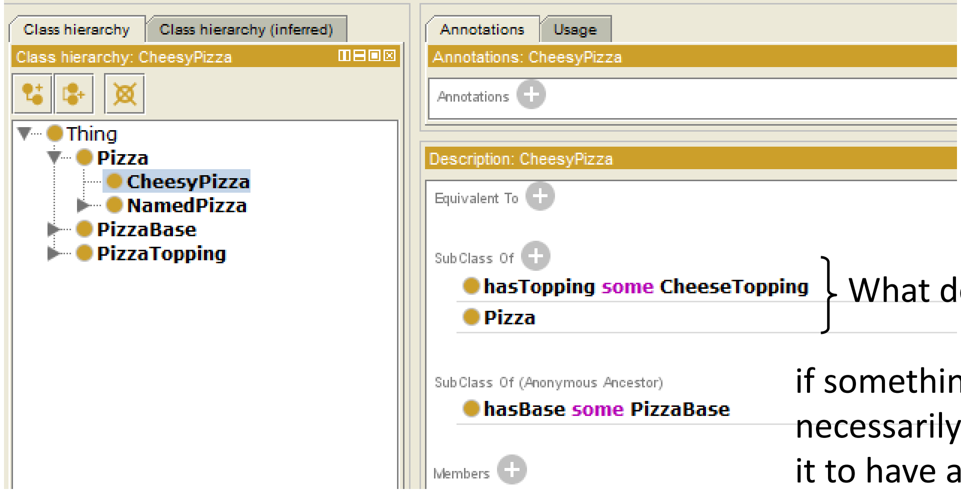
a Select the **MargheritaPizza** class


File
Edit
View
Reasoner
Tools
Refactor
Window
Help
Undo Ctrl-Z
Redo Ctrl+Maj-Z
Cut Ctrl-X
Copy Ctrl-C
Paste Ctrl-V
Delete ... Ctrl-Supprimer
Find in view... Ctrl-F
Create new Ctrl-N
Create child Ctrl-Barre oblique inverse
Create sibling Ctrl-Barre oblique
Duplicate selected class... Ctrl+Maj-C
Convert to primitive class
Convert to defined class
Add covering axiom
Make all individuals distinct...
Make primitive siblings disjoint
Remove disjoints for subclasses...

Select the **Make primitive siblings** disjoint option in the **Edit** menu b

Necessary and sufficient conditions

Create a subclass of **Pizza** called **CheesyPizza** and specify that it has at least one topping that is a kind of **CheeseTopping**



Class hierarchy | Class hierarchy (inferred)

Class hierarchy: CheesyPizza

- Thing
  - Pizza
    - CheesyPizza
    - NamedPizza
  - PizzaBase
  - PizzaTopping

Annotations | Usage

Annotations: CheesyPizza

Annotations

Description: CheesyPizza

Equivalent To

SubClass Of

- hasTopping some CheeseTopping
- Pizza

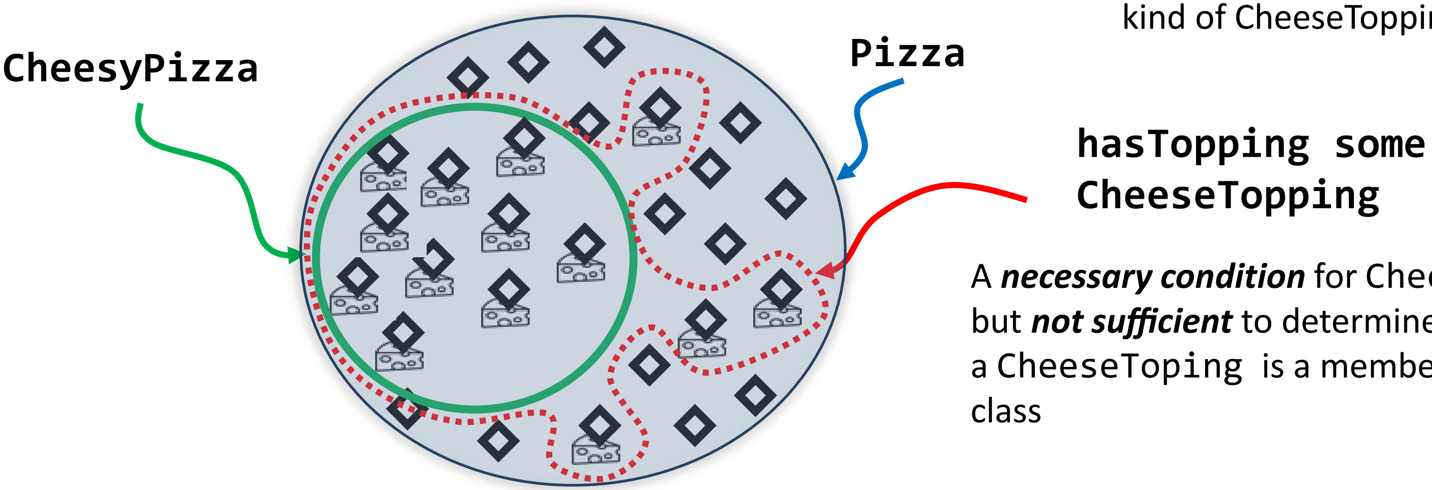
SubClass Of (Anonymous Ancestor)

- hasBase some PizzaBase

Members

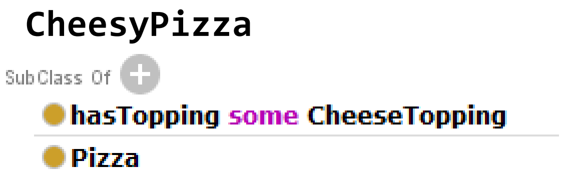
} What does it means ?

if something is a CheesyPizza it is necessarily a Pizza and it is necessary for it to have at least one topping that is a kind of CheeseTopping.



A ***necessary condition*** for CheesyPizza individuals but ***not sufficient*** to determine that an individual with a CheeseToping is a member of the CheesyPizza class

Necessary and sufficient conditions



CheesyPizza

SubClass Of

● hasTopping some CheeseTopping

● Pizza

Turtle

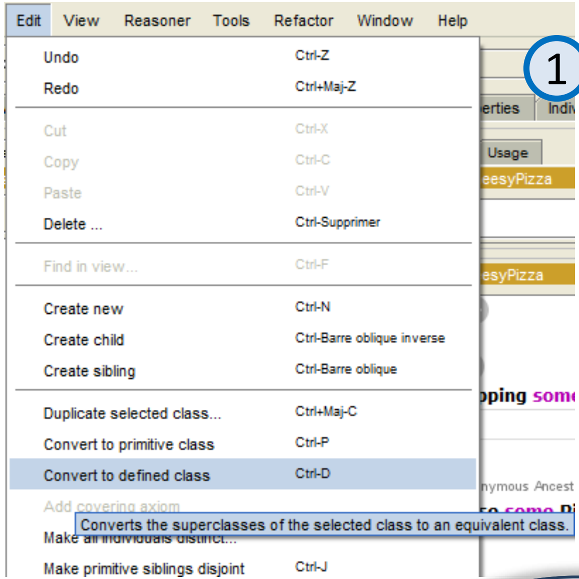
```
###  http://www.pizza.com/ontologies/pizza.owl#CheesyPizza

:CheesyPizza rdf:type owl:Class ;
             rdfs:subClassOf :Pizza ,
                             [ rdf:type owl:Restriction ;
                               owl:onProperty :hasTopping ;
                               owl:someValuesFrom :CheeseTopping
                             ] .
```

RDF/XML

```
<!-- http://www.pizza.com/ontologies/pizza.owl#CheesyPizza -->

    <owl:Class rdf:about="http://www.pizza.com/ontologies/pizza.owl#CheesyPizza">
        <rdfs:subClassOf rdf:resource="http://www.pizza.com/ontologies/pizza.owl#Pizza"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.pizza.com/ontologies/pizza.owl#hasTopping"/>
                <owl:someValuesFrom rdf:resource="http://www.pizza.com/ontologies/pizza.owl#CheeseTopping"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>
```

Necessary and sufficient conditions

Convert the *necessary* conditions for **CheesyPizza** into ***necessary & sufficient*** conditions



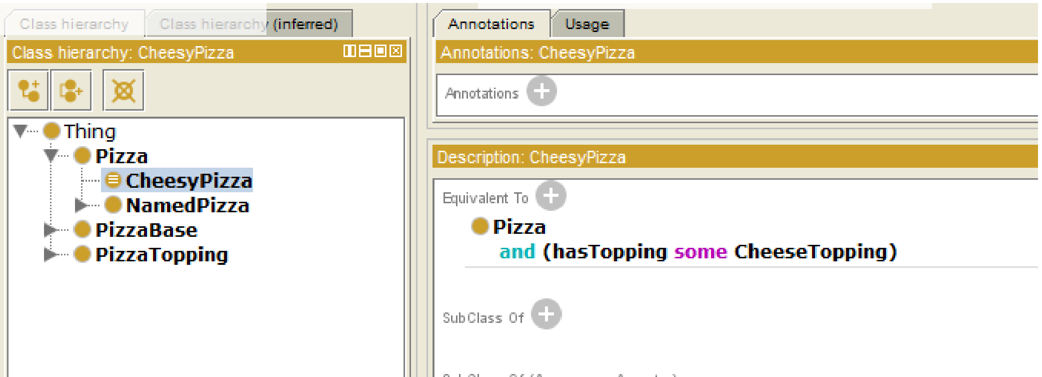
Edit View Reasoner Tools Refactor Window Help

```
Undo                          Ctrl-Z
Redo                          Ctrl+Maj-Z
Cut                           Ctrl-X
Copy                          Ctrl-C
Paste                         Ctrl-V
Delete ...                    Ctrl-Supprimer
Find in view...               Ctrl-F
Create new                    Ctrl-N
Create child                  Ctrl-Barre oblique inverse
Create sibling                Ctrl-Barre oblique
Duplicate selected class...   Ctrl+Maj-C
Convert to primitive class    Ctrl-P
Convert to defined class      Ctrl-D
Add covering axiom
Make all individuals distinct...
Make primitive siblings disjoint   Ctrl-J
```

Converts the superclasses of the selected class to an equivalent class.

1 Ensure that **CheesyPizza** is selected in the class hierarchy and then in the Edit menu select `Convert to defined class`

2 The Class Description View should now look like this



Class hierarchy: CheesyPizza

- Thing
  - Pizza
    - CheesyPizza
    - NamedPizza
  - PizzaBase
  - PizzaTopping

Annotations: CheesyPizza

Annotations

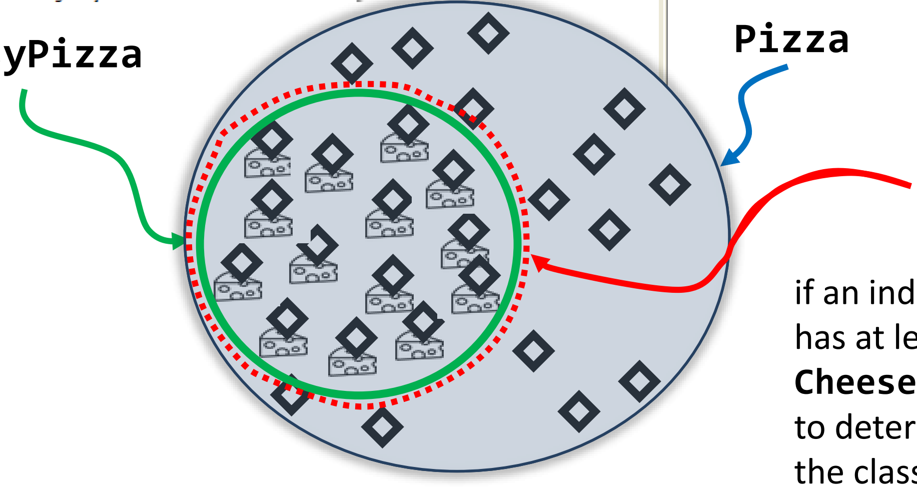
Description: CheesyPizza

Equivalent To
Pizza
and (hasTopping some CheeseTopping)

SubClass Of

SubClass Of (Anonymous Ancestor)
hasBase some PizzaBase

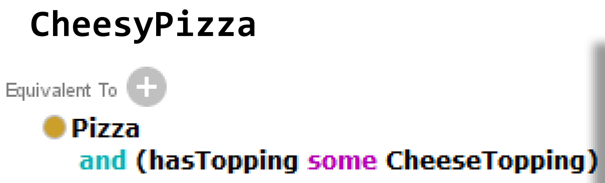
Members



hasTopping some CheeseTopping

if an individual is a member of the class **Pizza** and it has at least one topping that is a member of the class **CheeseTopping** then these conditions are sufficient to determine that the individual ***must*** be a member of the class **CheesyPizza**

Necessary and sufficient conditions



CheesyPizza

Pizza
and (hasTopping some CheeseTopping)

Turtle

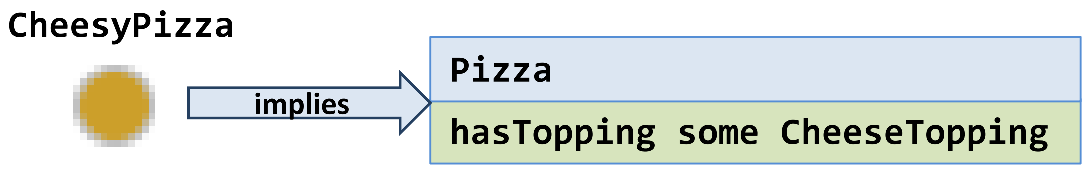
```
###  http://www.pizza.com/ontologies/pizza.owl#CheesyPizza

:CheesyPizza rdf:type owl:Class ;
             owl:equivalentClass [ rdf:type owl:Class ;
                                   owl:intersectionOf (
                                       :Pizza
                                       [ rdf:type owl:Restriction ;
                                         owl:onProperty :hasTopping ;
                                         owl:someValuesFrom :CheeseTopping
                                       ]
                                     )
                                 ] .
```

RDF/XML

```
<!-- http://www.pizza.com/ontologies/pizza.owl#CheesyPizza -->

<owl:Class rdf:about="http://www.pizza.com/ontologies/pizza.owl#CheesyPizza">
     <owl:equivalentClass>
         <owl:Class>
             <owl:intersectionOf rdf:parseType="Collection">
                 <rdf:Description rdf:about="http://www.pizza.com/ontologies/pizza.owl#Pizza"/>
                 <owl:Restriction>
                     <owl:onProperty rdf:resource="http://www.pizza.com/ontologies/pizza.owl#hasTopping"/>
                     <owl:someValuesFrom
                           rdf:resource="http://www.pizza.com/ontologies/pizza.owl#CheeseTopping"/>
                 </owl:Restriction>
             </owl:intersectionOf>
         </owl:Class>
     </owl:equivalentClass>
</owl:Class>
```

Primitive and Defined Classes

Necessary Conditions

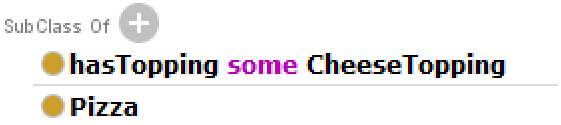


CheesyPizza

implies

Pizza

hasTopping some CheeseTopping



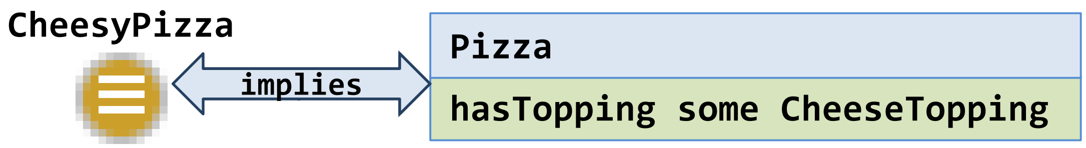
● hasTopping some CheeseTopping

● Pizza

Primitive Class

If an individual is a member of **CheesyPizza** then it must satisfy the conditions. However if some individual satisfies these necessary conditions, we cannot say that it is a member of **CheesyPizza**

Necessary & Sufficient Conditions

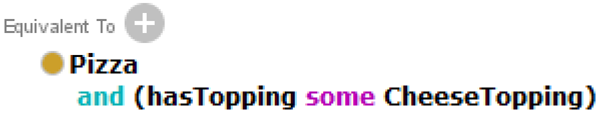


CheesyPizza

implies

Pizza

hasTopping some CheeseTopping



● Pizza
and (hasTopping some CheeseTopping)

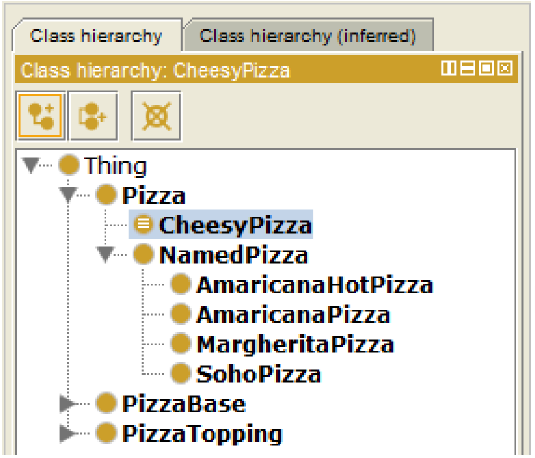
*Defined Class**

If an individual is a member of **CheesyPizza** then it must satisfy the conditions. If some individual satisfies the conditions then the individual must be a member of **CheesyPizza**

** Classes that have at least one set of necessary and sufficient conditions are known as **defined** classes — they have a definition, and any individual that satisfies the definition will belong to the class.*

Automated Classification of Defined Classes

Use the reasoner to automatically compute the subclasses of **CheesyPizza** (select **Start reasoner** or **Synchronize reasoner** in the **Reasoner** menu).



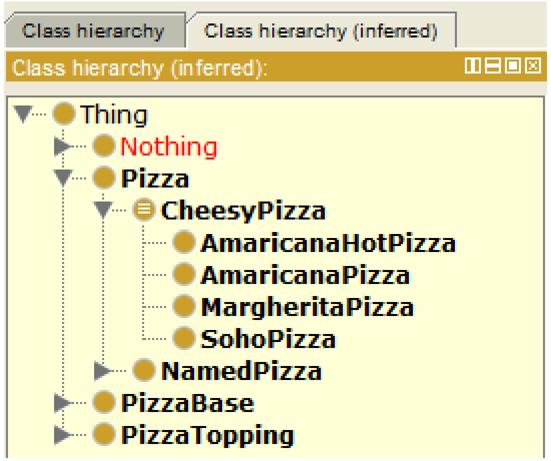
Class hierarchy | Class hierarchy (inferred)

Class hierarchy: CheesyPizza

- Thing
  - Pizza
    - CheesyPizza
    - NamedPizza
      - AmaricanaHotPizza
      - AmaricanaPizza
      - MargheritaPizza
      - SohoPizza
  - PizzaBase
  - PizzaTopping

Asserted Class Hierarchy

reasoner

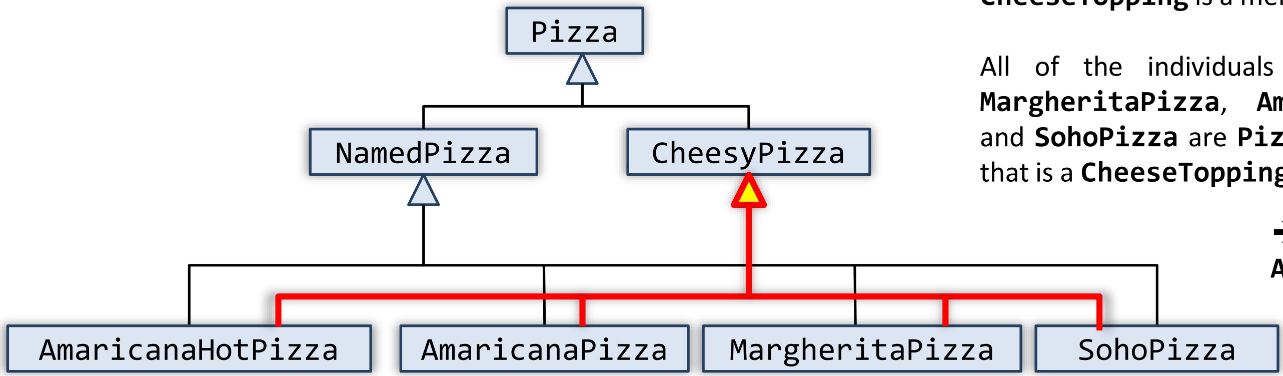


Class hierarchy | Class hierarchy (inferred)

Class hierarchy (inferred):

- Thing
  - Nothing
  - Pizza
    - CheesyPizza
      - AmaricanaHotPizza
      - AmaricanaPizza
      - MargheritaPizza
      - SohoPizza
    - NamedPizza
  - PizzaBase
  - PizzaTopping

Inferred Class Hierarchy



Any individual that is a **Pizza** and has at least one topping that is a **CheeseTopping** is a member of the class **CheesyPizza**

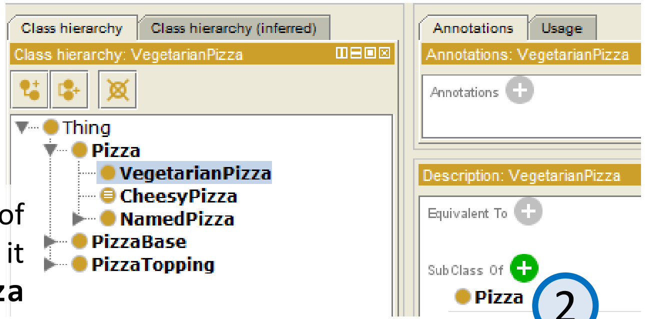
All of the individuals that are described by the classes **MargheritaPizza**, **AmericanaPizza**, **AmericanHotPizza** and **SohoPizza** are **Pizzas** and they have at least one topping that is a **CheeseTopping**

➔ **MargheritaPizza**, **AmericanaPizza**, **AmericanHotPizza** and **SohoPizza** must be subclasses of **CheesyPizza**

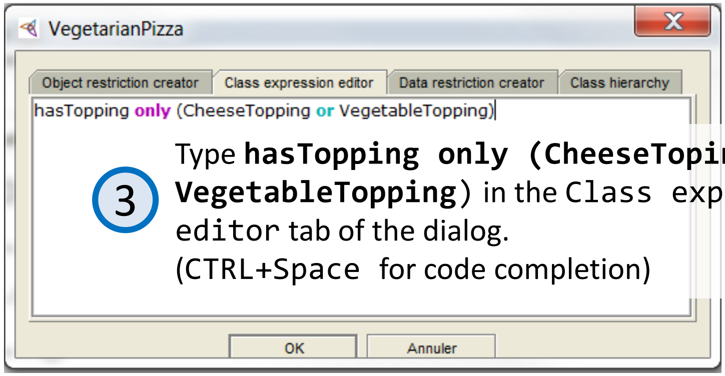
Creating a class with an universal restriction

Create a class to describe a **VegetarianPizza**, a class whose members can ***only*** have toppings that are **CheeseTopping** or **VegetableTopping**.

1 Create a subclass of **Pizza**, and name it **VegetarianPizza**



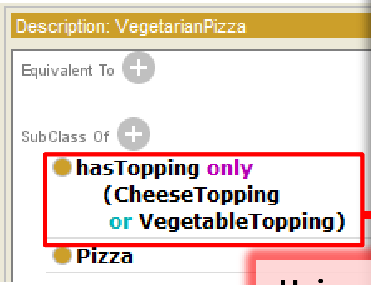
2 Click on the **Add SubClass of** button on the **VegetarianPizza** class Description View.



hasTopping only (CheeseTopping or VegetableTopping)

3 Type **hasTopping only (CheeseToping or VegetableTopping)** in the Class expression editor tab of the dialog. (CTRL+Space for code completion)

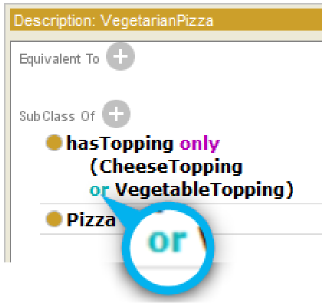
4 Validate and ensure that **VegetarianPizza** description is correct


Description: VegetarianPizza
Equivalent To
SubClass Of
hasTopping only (CheeseTopping or VegetableTopping)
Pizza

Universal (∀ quantifier)restriction

```
###  http://www.pizza.com/ontologies/pizza.owl#VegetarianPizza
:VegetarianPizza
       rdf:type owl:Class ;
       rdfs:subClassOf :Pizza ,
                         [ rdf:type owl:Restriction ;
                           owl:onProperty :hasTopping ;
                           owl:allValuesFrom [ rdf:type owl:Class ;
                                               owl:unionOf (
                                                    :CheeseTopping
                                                    :VegetableTopping
                                               )
                                             ]
                          ] .
```

Interpretation of universal restrictions


Description: VegetarianPizza

Equivalent To

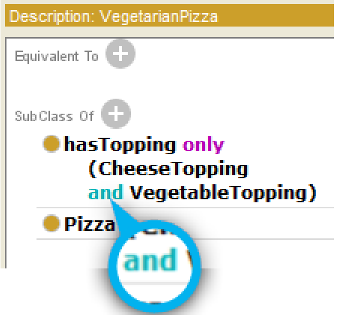
SubClass Of

- hasTopping only (CheeseTopping or VegetableTopping)
- Pizza

```
###  http://www.pizza.com/ontologies/pizza.owl#VegetarianPizza
:VegetarianPizza
      rdf:type owl:Class ;
      rdfs:subClassOf :Pizza ,
                       [ rdf:type owl:Restriction ;
                         owl:onProperty :hasTopping ;
                         owl:allValuesFrom [ rdf:type owl:Class ;
                                             owl:unionOf (
                                                 :CheeseTopping
                                                 :VegetableTopping
                                             )
                                           ]
                        ] .
```

If something is a member of the class **VegetarianPizza** it is necessary for it to be a kind of **Pizza** and it is necessary for it to *only* (∀ universal quantifier) have toppings that are kinds of **CheeseTopping** **or** kinds of **VegetableTopping**.

and


Description: VegetarianPizza

Equivalent To

SubClass Of

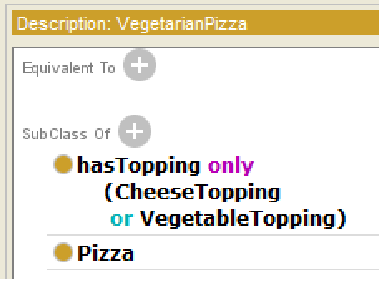
- hasTopping only (CheeseTopping and VegetableTopping)
- Pizza

```
:VegetarianPizza
       rdf:type owl:Class ;
       rdfs:subClassOf :Pizza ,
                        [ rdf:type owl:Restriction ;
                          owl:onProperty :hasTopping ;
                          owl:allValuesFrom [ rdf:type owl:Class ;
                                              owl:intersectionOf (
                                                  :CheeseTopping
                                                  :VegetableTopping
                                              )
                                            ]
                         ] .
```

If something is a member of the class **VegetarianPizza** it is necessary for it to be a kind of **Pizza** and it is necessary for it to *only* (∀ universal quantifier) have toppings that are kinds of **CheeseTopping** **and** kinds of **VegetableTopping**.

⚠ Inconsistent because **CheeseTopping** and **VegetableTopping** are disjoint classes

Interpretation of universal restrictions



Description: VegetarianPizza

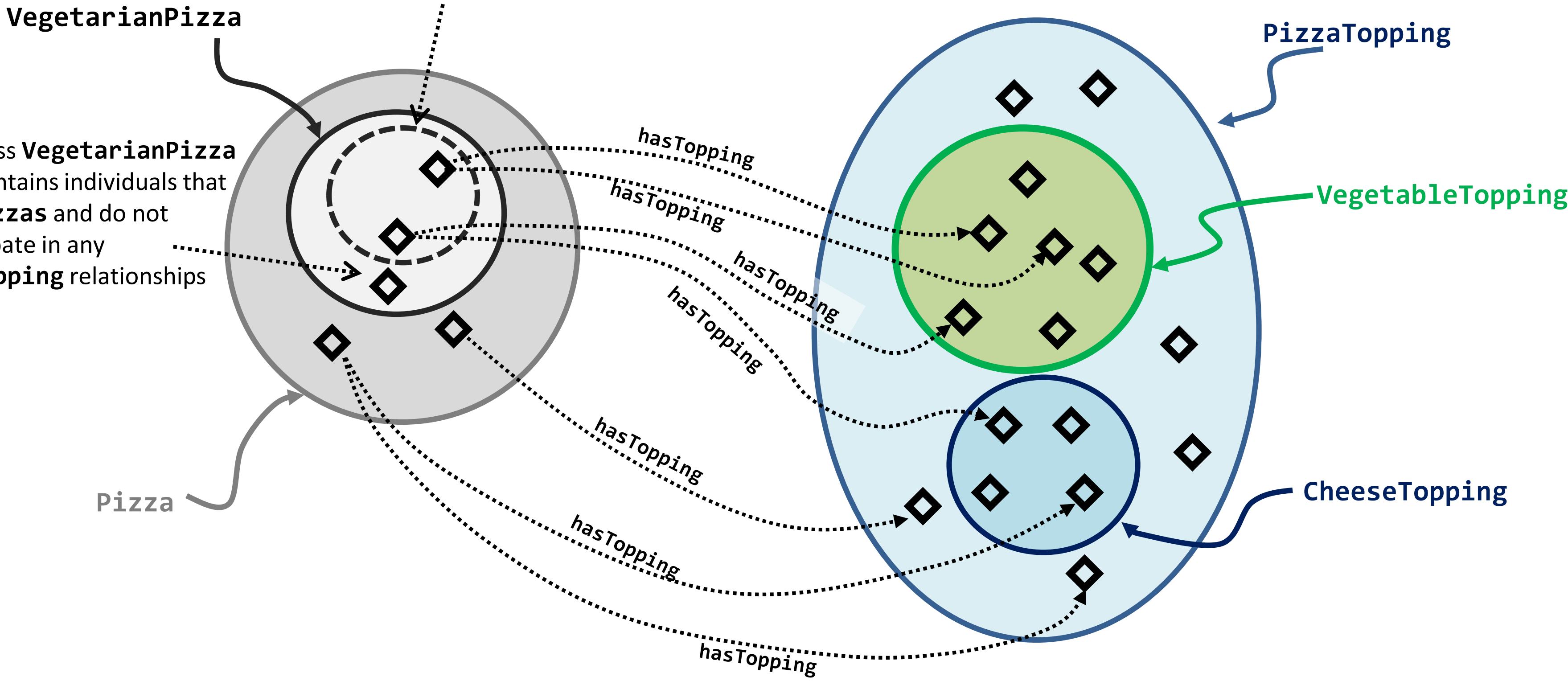
Equivalent To

SubClass Of

- hasTopping only (CheeseTopping or VegetableTopping)
- Pizza

est-ce vrai si on a fait une defined class ? Pizza sans topping classée dans Vegy ?

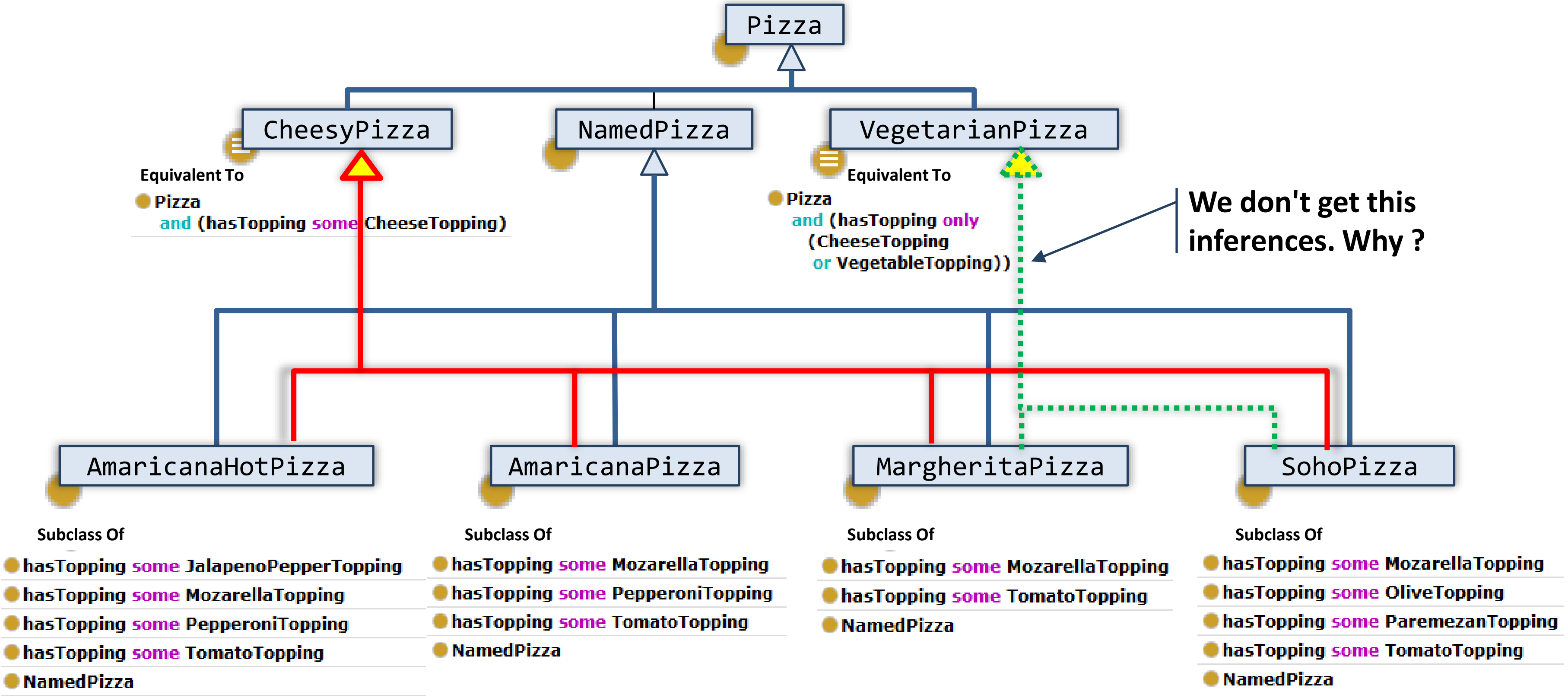
All **hasTopping** relationships that individuals which are members of the class **VegetarianPizza** participate in must be to individuals that are either members of the class **CheeseTopping** or **VegetableTopping**



The class **VegetarianPizza** also contains individuals that are **Pizzas** and do not participate in any **hasTopping** relationships

Classification of NamedPizzas

Use the reasoner to classify the ontology (**Start Reasoner** or **Synchronize Reasoner** button in the **Reasoner** drop down menu)



CheesyPizza — Equivalent To
- Pizza and (hasTopping some CheeseTopping)

VegetarianPizza — Equivalent To
- Pizza and (hasTopping only (CheeseTopping or VegetableTopping))

We don't get this inferences. Why ?

AmaricanaHotPizza — Subclass Of
- hasTopping some JalapenoPepperTopping
- hasTopping some MozarellaTopping
- hasTopping some PepperoniTopping
- hasTopping some TomatoTopping
- NamedPizza

AmaricanaPizza — Subclass Of
- hasTopping some MozarellaTopping
- hasTopping some PepperoniTopping
- hasTopping some TomatoTopping
- NamedPizza

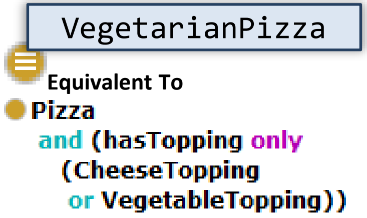
MargheritaPizza — Subclass Of
- hasTopping some MozarellaTopping
- hasTopping some TomatoTopping
- NamedPizza

SohoPizza — Subclass Of
- hasTopping some MozarellaTopping
- hasTopping some OliveTopping
- hasTopping some ParemezanTopping
- hasTopping some TomatoTopping
- NamedPizza

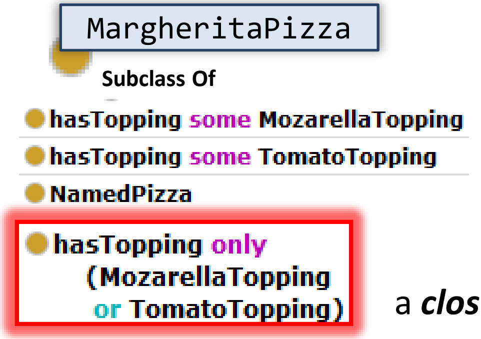
MargheritaPizza and **SohoPizza** have something missing from their definition that means they cannot be classified as subclasses of **VegetarianPizza**

Open World Assumption (OWA)

- **Open World Assumption** : we cannot assume something doesn’t exist until it is explicitly stated that it does not exist
  - In other words, because something hasn’t been stated to be true, it cannot be assumed to be false — it is assumed that ‘*the knowledge just hasn’t been added to the knowledge base*’.



```
VegetarianPizza
  Equivalent To
  Pizza
    and (hasTopping only
      (CheeseTopping
       or VegetableTopping))
```



```
MargheritaPizza
  Subclass Of
  hasTopping some MozarellaTopping
  hasTopping some TomatoTopping
  NamedPizza
  hasTopping only
      (MozarellaTopping
       or TomatoTopping)
```

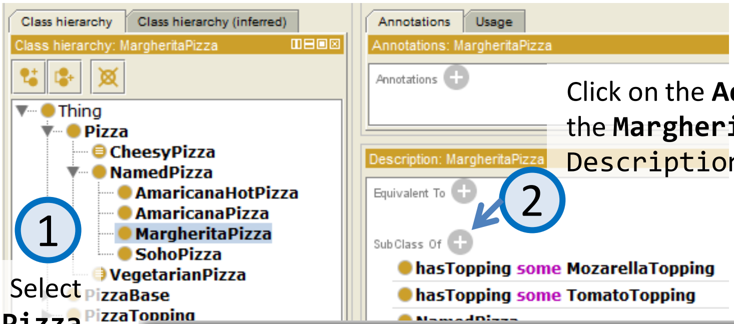
OWA → until we explicitly say that a **`MargheritaPizza`** ***only*** has these kinds of toppings, it is assumed (by the reasoner) that a **`MargheritaPizza`** could have other toppings

a ***closure axiom*** must be added on the hasTopping property

- **Closure axiom** on a property : a universal restriction (***only***) that acts along the property to say that it can only be filled by the specified fillers.

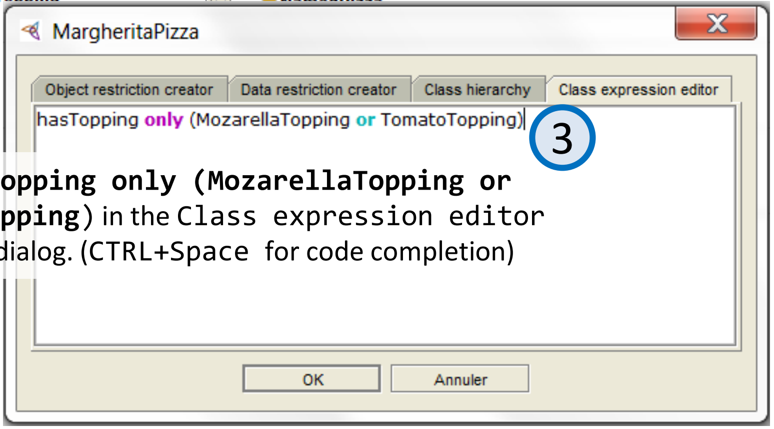
  restriction filler : the **union** of the fillers that occur in the existential restrictions for the property

Adding a closure axiom to MargheritaPizza



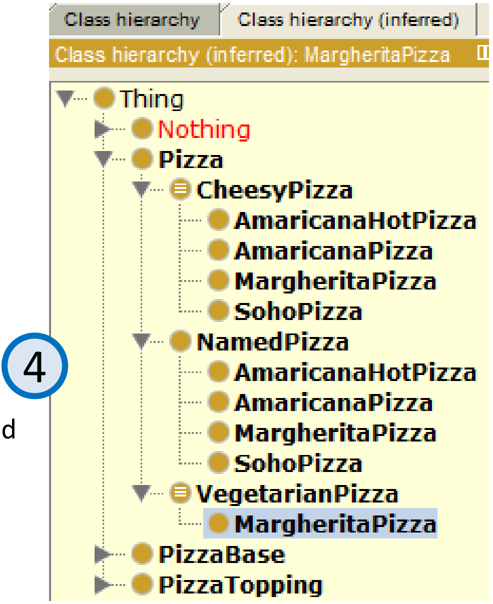
1 Select **MargheritaPizza**

2 Click on the **Add SubClass of** button on the **MargheritaPizza** class Description View.



hasTopping only (MozarellaTopping or TomatoTopping)

3 Type **hasTopping only (MozarellaTopping or TomatoTopping)** in the Class expression editor tab of the dialog. (CTRL+Space for code completion)

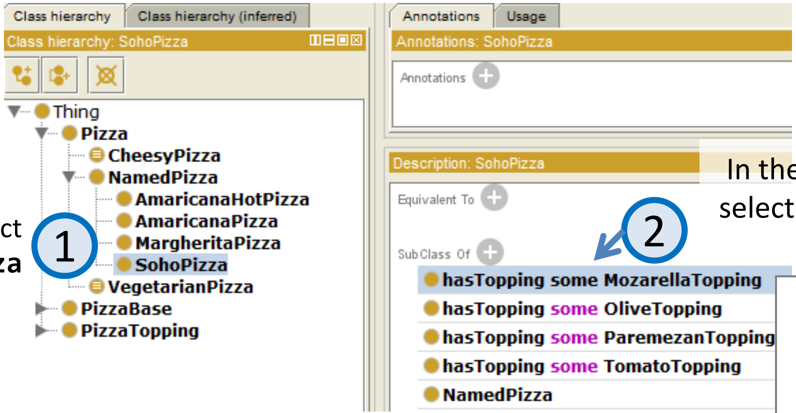


4 Execute the Reasoner to verify that **MargheritaPizza** is correctly classified

Adding a closure axiom to other NamedPizzas

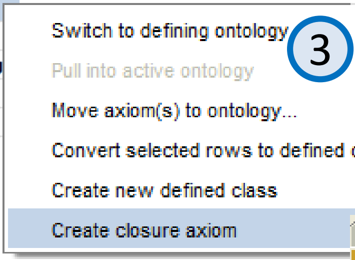
Add a closure axiom on the hasTopping property for **SohoPizza** .

1 Select **SohoPizza**



- hasTopping some MozarellaTopping
- hasTopping some OliveTopping
- hasTopping some ParemezanTopping
- hasTopping some TomatoTopping
- NamedPizza

2 In the class description view, select one of the restrictions


Switch to defining ontology
Pull into active ontology
Move axiom(s) to ontology...
Convert selected rows to defined class
Create new defined class
Create closure axiom

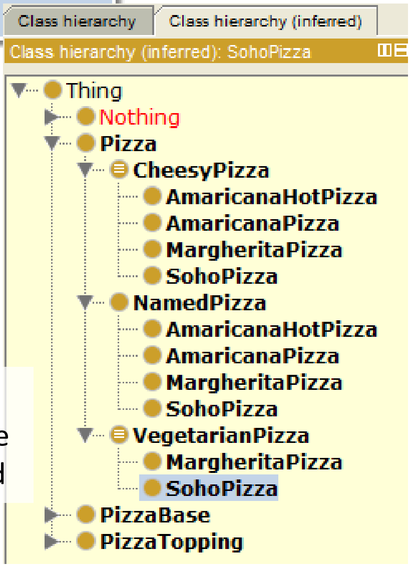
3 Right click the restriction and select `Create closure axiom`.

Type **`hasTopping only (MozarellaTopping or TomatoTopping`**) in the `Class expression editor` tab of the dialog. (`CTRL+Space` for code completion)

4 Do the same for **`AmericanaPizza`** and **`AmericanaHotPizza`**

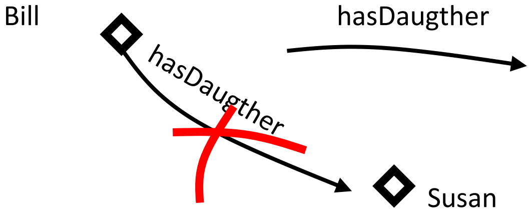
5 Execute the reasoner

6 verify that **`NamedPizzas`** are correctly classified



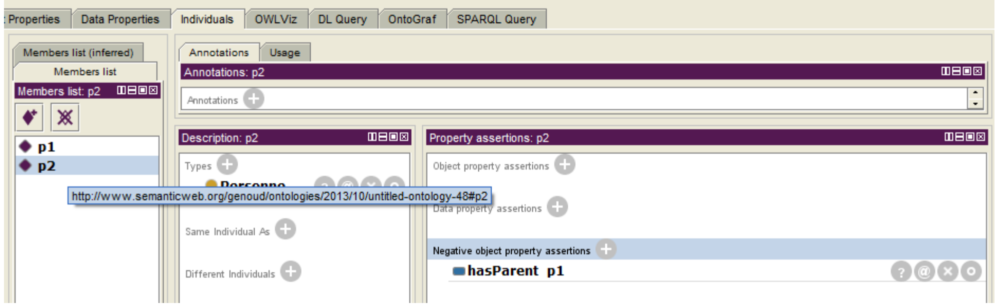
Negative properties assertions

- We can state that two individuals are *not* connected by a property.



```
[]  rdf:type owl:NegativePropertyAssertion ;
    owl:sourceIndividual   :Bill ;
    owl:assertionProperty  :hasDaughter ;
    owl:targetIndividual   :Susan .
```

→ to make statements where we know **something that is not true**.
This kind of information is particularly important in OWL where the default stance is that anything is possible until you say otherwise (OWA).



http://www.semanticweb.org/genoud/ontologies/2013/10/untitled-ontology-48#p2

hasParent p1

Value Partition

- we want to express the spiciness that can be one of the three values : Mild, Medium and Hot

  →use a **value partition**

- **Value Partition**:
  - restrict the range of possible values to an exhaustive list
  - not part of OWL
  - **a design pattern** : a solution that has been developed by experts and is now recognized as a proven solution for solving common modelling problems

Creating a Value Partition in OWL

1. Create a class to represent the ValuePartition.

   `SpicinessValuePartition` to represent a ‘spiciness’ ValuePartition

2. Create subclasses of the ValuePartition to represent the possible options for the ValuePartition.

   `Mild`, `Medium` and `Hot` classes as subclasses `SpicinessValuePartition`.

3. Make the subclasses of the ValuePartition class disjoint.
4. Provide a *covering axiom* to make the list of value types exhaustive
5. Create an object property for the ValuePartition.

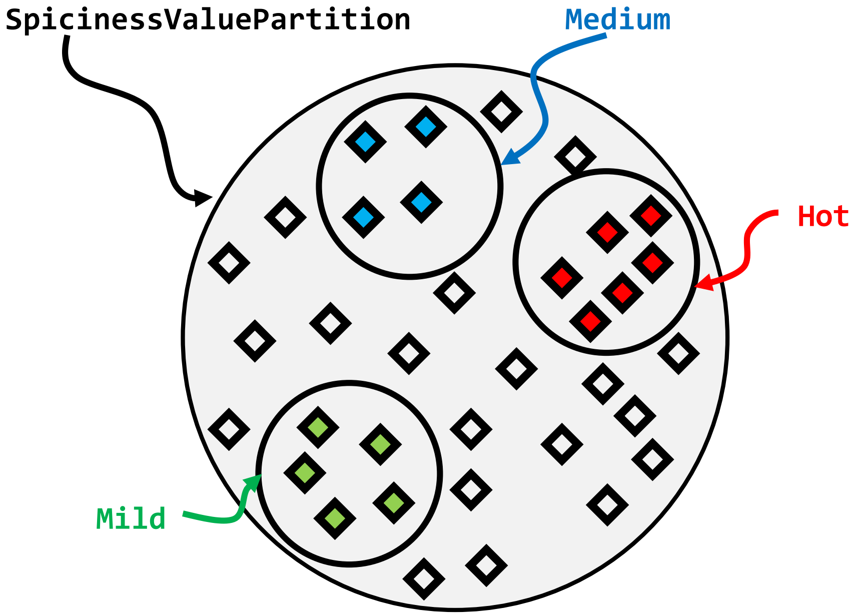
   `hasSpiciness` property

6. Make the property functional.
7. Set the range of the property as the ValuePartition class.

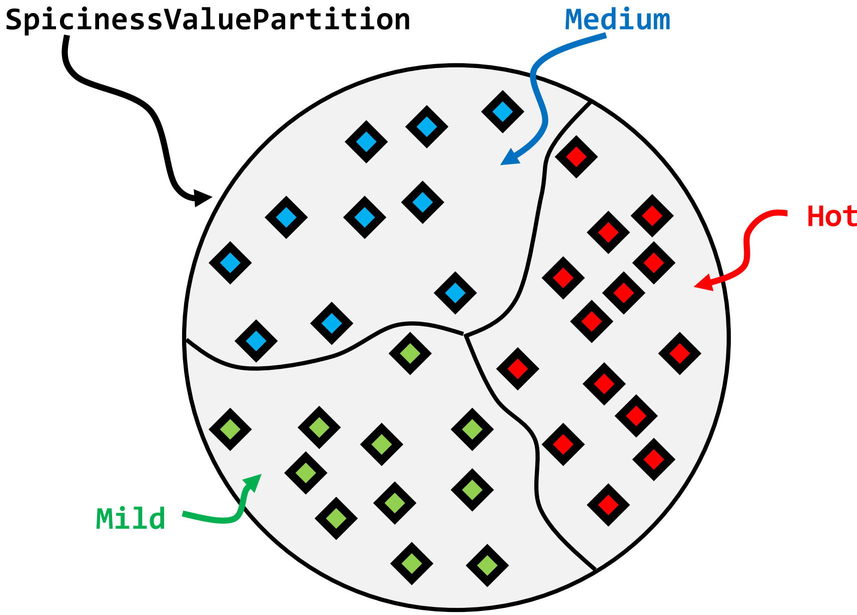
   set the range of `hasSpiciness` property to `SpicinessValuePartition`.

Covering Axioms

- A covering axiom consists of two parts:
  - the class that is being ‘covered’,
  - and the classes that form the covering
- in OWL → define the union of the classes forming the covering as a superclass of the covered class



without covering axiom



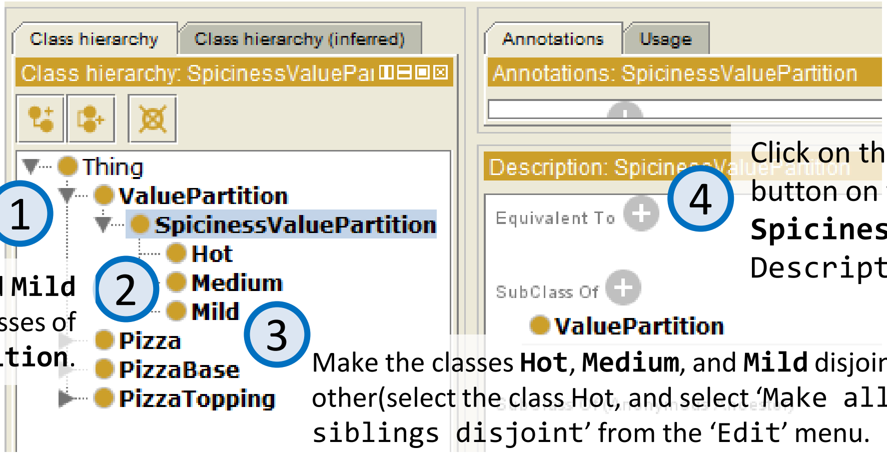
with covering axiom

Mild, **Medium** and **Hot** are subclasses of **SpicinessValuePartition** and **Mild** U **Medium** U **Hot** is a superclass of **SpicinessValuePartition**

Creating SpicinessValuePartition

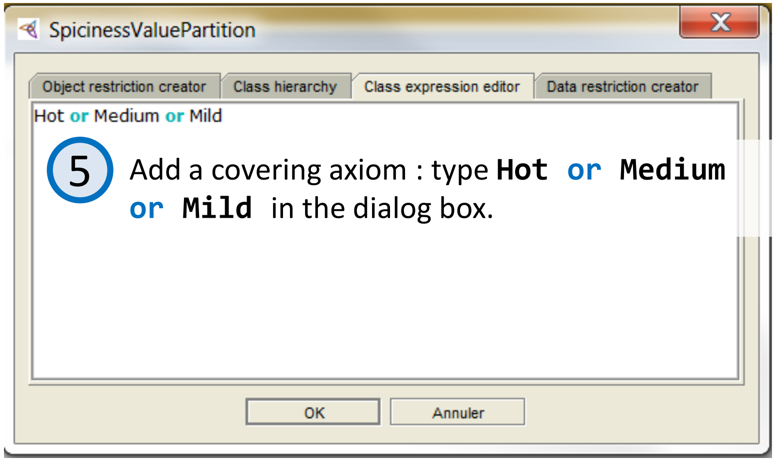
1 Create **ValuePartition** a sub class of **Thing** and **SpicinessValuePartition** a sub class of **ValuePartition**.

2 Create **Hot**, **Medium**, and **Mild** three subclasses of **SpicinessValuePartition**.



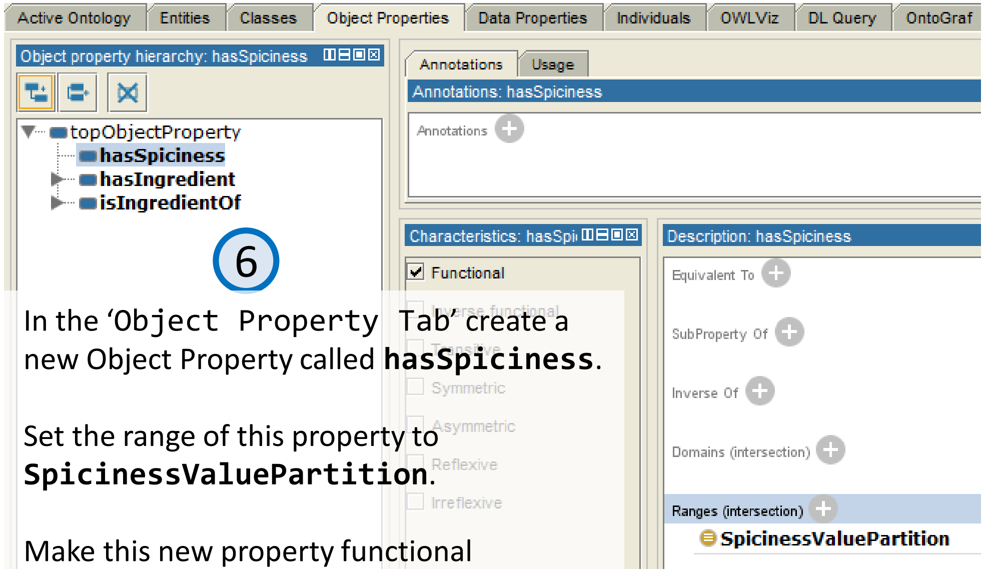
3 Make the classes **Hot**, **Medium**, and **Mild** disjoint from each other(select the class Hot, and select 'Make all primitive siblings disjoint' from the 'Edit' menu.

4 Click on the **Add Equivalent To** button on the **SpicinessValuePartition** class Description View.



Hot or Medium or Mild

5 Add a covering axiom : type **Hot or Medium or Mild** in the dialog box.



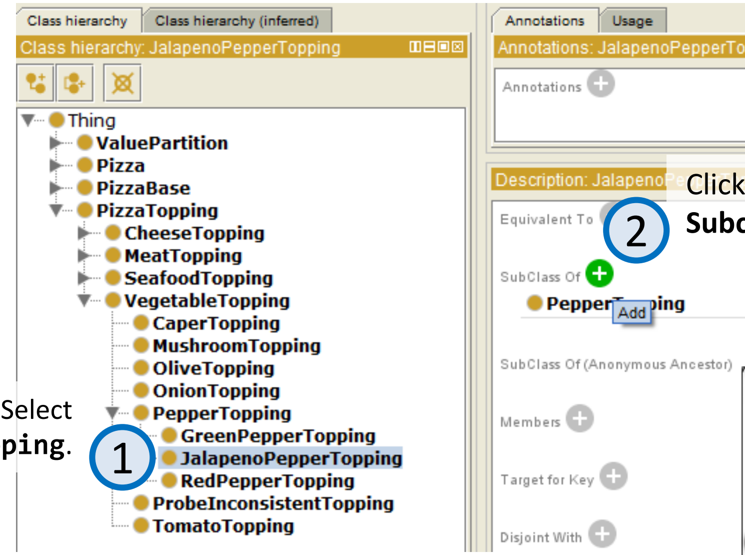
☑ Functional
□ Inverse functional
□ Transitive
□ Symmetric
□ Asymmetric
□ Reflexive
□ Irreflexive

6 In the 'Object Property Tab' create a new Object Property called **hasSpiciness**.

Set the range of this property to **SpicinessValuePartition**.

Make this new property functional

Adding Spiciness to Pizza Toppings

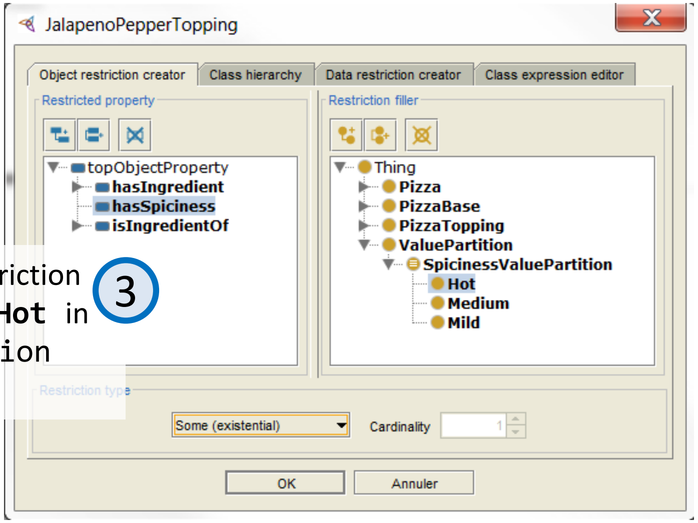

Class hierarchy
Class hierarchy (inferred)
Class hierarchy: JalapenoPepperTopping
Annotations
Usage
Annotations: JalapenoPepperTop
Annotations
Description: JalapenoPe
Equivalent To
SubClass Of
Add
SubClass Of (Anonymous Ancestor)
Members
Target for Key
Disjoint With

- Thing
  - ValuePartition
  - Pizza
  - PizzaBase
  - PizzaTopping
    - CheeseTopping
    - MeatTopping
    - SeafoodTopping
    - VegetableTopping
      - CaperTopping
      - MushroomTopping
      - OliveTopping
      - OnionTopping
      - PepperTopping
        - GreenPepperTopping
        - JalapenoPepperTopping
        - RedPepperTopping
      - ProbeInconsistentTopping
      - TomatoTopping

SubClass Of: PepperTopping

1 Select `JalapenoPepperTopping`.

2 Click on the **Add Subclass Of** button

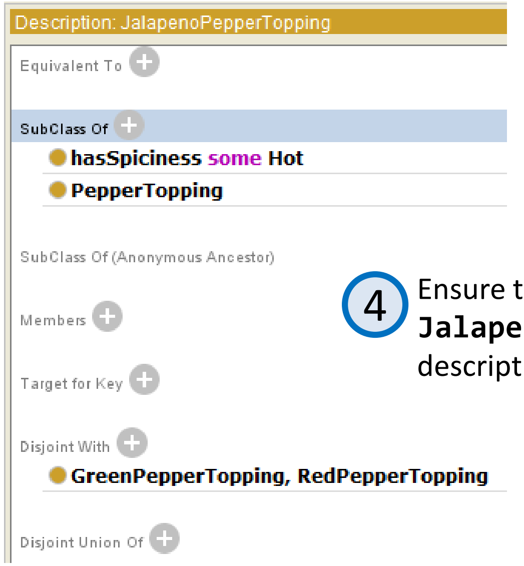


Restricted property:
- topObjectProperty
  - hasIngredient
  - hasSpiciness
  - isIngredientOf

Restriction filler:
- Thing
  - Pizza
  - PizzaBase
  - PizzaTopping
  - ValuePartition
    - SpicinessValuePartition
      - Hot
      - Medium
      - Mild

Restriction type: Some (existential)

3 Create an existential restriction **hasSpiciness some Hot** in the 'Object restriction creator' dialog



SubClass Of:
- hasSpiciness some Hot
- PepperTopping

Disjoint With:
- GreenPepperTopping, RedPepperTopping

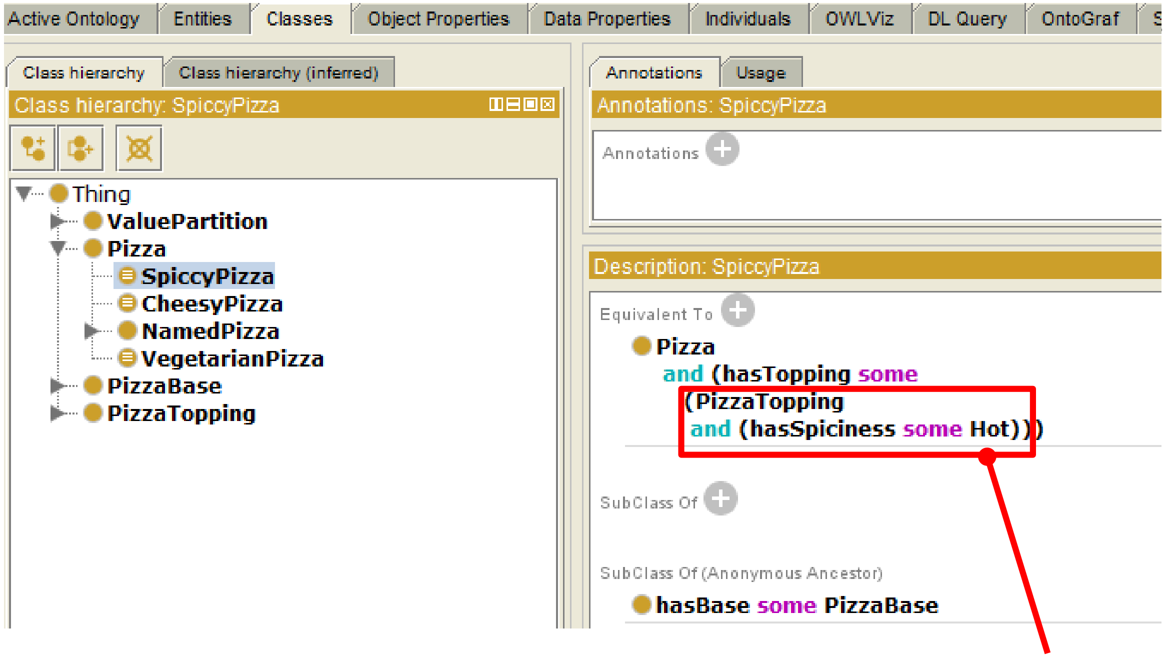
4 Ensure that **JalapenoPepperTopping** description looks like this

5 *Optional*

Repeat this for each of the bottom level PizzaToppings (those that have no subclasses) to state it's spiciness (one of Hot, Medium or Mild)

Creating SpicyPizza as subclass of Pizza

Create **SpiccyPizza** as subclass of **Pizza** with the following



- Thing
  - ValuePartition
  - Pizza
    - SpiccyPizza
    - CheesyPizza
    - NamedPizza
    - VegetarianPizza
  - PizzaBase
  - PizzaTopping

Equivalent To

```
Pizza
 and (hasTopping some
  (PizzaTopping
   and (hasSpiciness some Hot)))
```

SubClass Of (Anonymous Ancestor)

```
hasBase some PizzaBase
```

An anonymous class which contains the individuals that are members of the class **PizzaTopping** and also members of the class of individuals that are related to the members of class **Hot** via the **hasSpiciness** property
⇔ the things that are **PizzaToppings** and have a spiciness that is **Hot**.

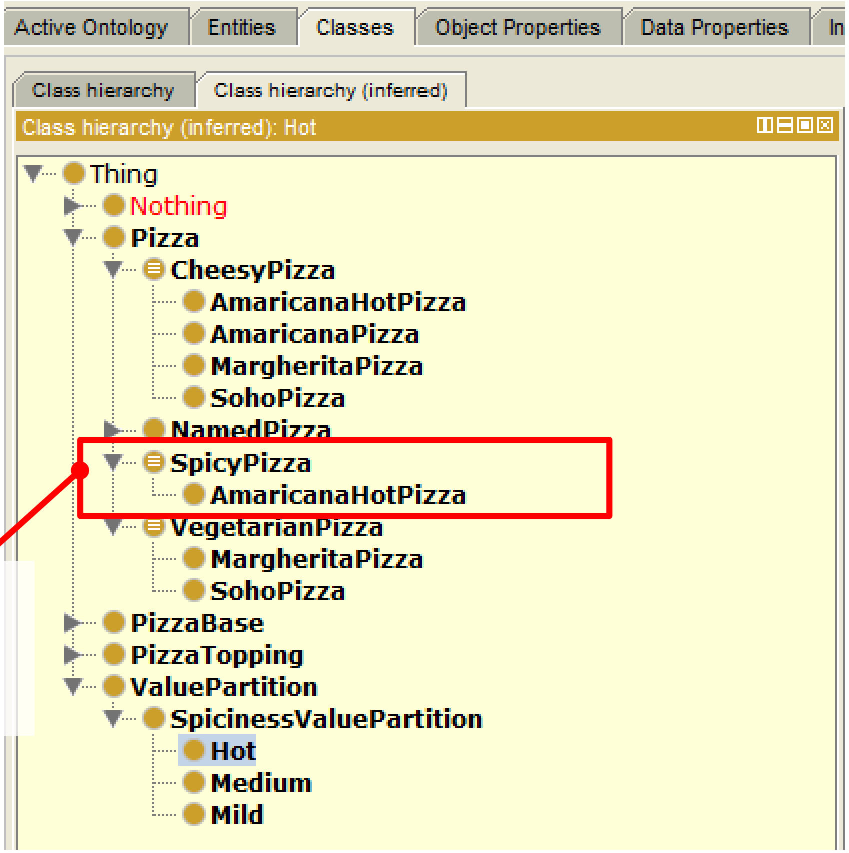
Meaning of **SpicyPizza** description :

- all members of **SpicyPizza** are **Pizzas** and have at least one topping that has a **Spiciness of Hot**
- anything that is a **Pizza** and has at least one topping that has a spiciness of **Hot** is a **SpicyPizza**

Classifying the ontology

1 Run the reasoner

2 Verify that **AmericanHotPizza** has been classified as a subclass of **SpicyPizza**

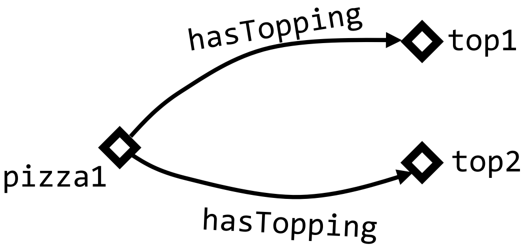


the reasoner has automatically computed that any individual that is a member of **AmericanHotPizza** is also a member of **SpicyPizza**

Cardinality Restrictions

- **Cardinality Restrictions**
  - describe the class of individuals that have at least, at most or exactly a specified number of relationships with other individuals or datatype values.
  - For a given property **P**,
    - Minimum Cardinality Restriction → the minimum number of **P** relationships that an individual must participate in.
    - Maximum Cardinality Restriction → the maximum number of **P** relationships that an individual can participate in.
    - Cardinality Restriction specifies the exact number of **P** relationships that an individual must participate in.

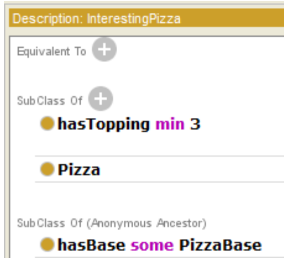
⚠ Relationships are only counted as separate relationships if it can be determined that the individuals that are the *fillers* for the relationships are *different* to each other.



The individual **`pizza1`** satisfies a *minimum cardinality restriction of 2* along the ***`hasTopping`*** property if the individuals **`top1`** and **`top2`** are distinct individuals

Creating and classifying a class with a cardinality restriction

1. Create a subclass of **Pizza** called **InterestingPizza**.
2. Press the Add button on the '*SubClass Of*' section of the class description view.
3. In the class expression editor type
    1. **hasTopping** as a property to be restricted.
    2. **min** to create a minimum cardinality restriction.
    3. **3** to specify a minimum cardinality of three
4. Press '*Enter*' to close the dialog and create the restriction.


Description: InterestingPizza

Equivalent To

SubClass Of
- hasTopping min 3
- Pizza

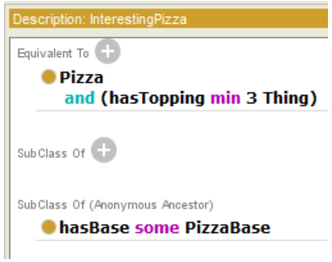
SubClass Of (Anonymous Ancestor)
- hasBase some PizzaBase

class description after step 4

5. Select the '*Convert to defined class*' option in the '*Edit*' menu.

What does this mean?

InterestingPizza : the set of individuals that are members of the class **Pizza** and that have at least three ***hasTopping*** relationships with other (distinct) individuals.


Description: InterestingPizza

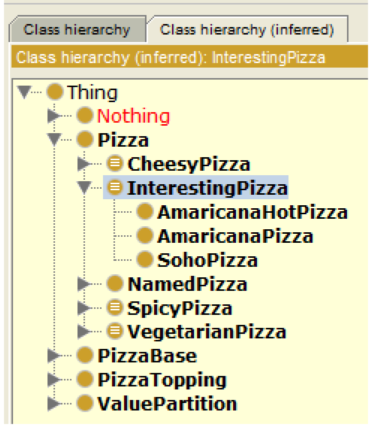
Equivalent To
- Pizza
  and (hasTopping min 3 Thing)

SubClass Of

SubClass Of (Anonymous Ancestor)
- hasBase some PizzaBase

class description after step 5

6. Run the reasoner


Class hierarchy | Class hierarchy (inferred)

Class hierarchy (inferred): InterestingPizza

```
Thing
  Nothing
  Pizza
    CheesyPizza
    InterestingPizza
      AmaricanaHotPizza
      AmaricanaPizza
      SohoPizza
    NamedPizza
    SpicyPizza
    VegetarianPizza
  PizzaBase
  PizzaTopping
  ValuePartition
```

class hierarchy after classification

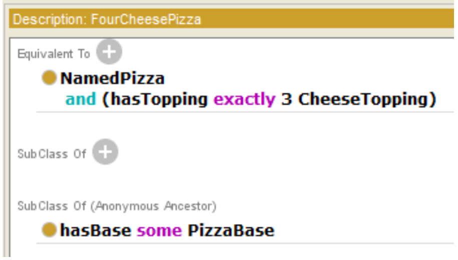
Qualified Cardinality Restrictions

- **Qualified Cardinality Restrictions**
  - more specific than cardinality restrictions → *they state the class of objects* within the restriction.

define a **FourCheesePizza** class that describes the set of individuals that are members of the class **NamedPizza** and that have exactly four **hasTopping** relationships with (distinct) individuals of the **CheeseTopping** class.

1. Create a subclass of **NamedPizza** called **FourCheesePizza**.
2. Press the Add button on the '*SubClass Of*' section of the class description view.
3. In the class expression editor type
   1. **hasTopping** as a property to be restricted.
   2. **exactly** to create an exact cardinality restriction.
   3. **4** to specify exact cardinality of four
   4. **CheeseTopping** to specify the type of topping
4. Press '*Enter*' to close the dialog and create the restriction.
5. Select the '*Convert to defined class*' option in the '*Edit*' menu.

to perform these steps it's also possible to use the *Object Restriction creator* tab in the dialog



Description: FourCheesePizza

Equivalent To

NamedPizza and (hasTopping exactly 3 CheeseTopping)

SubClass Of

SubClass Of (Anonymous Ancestor)

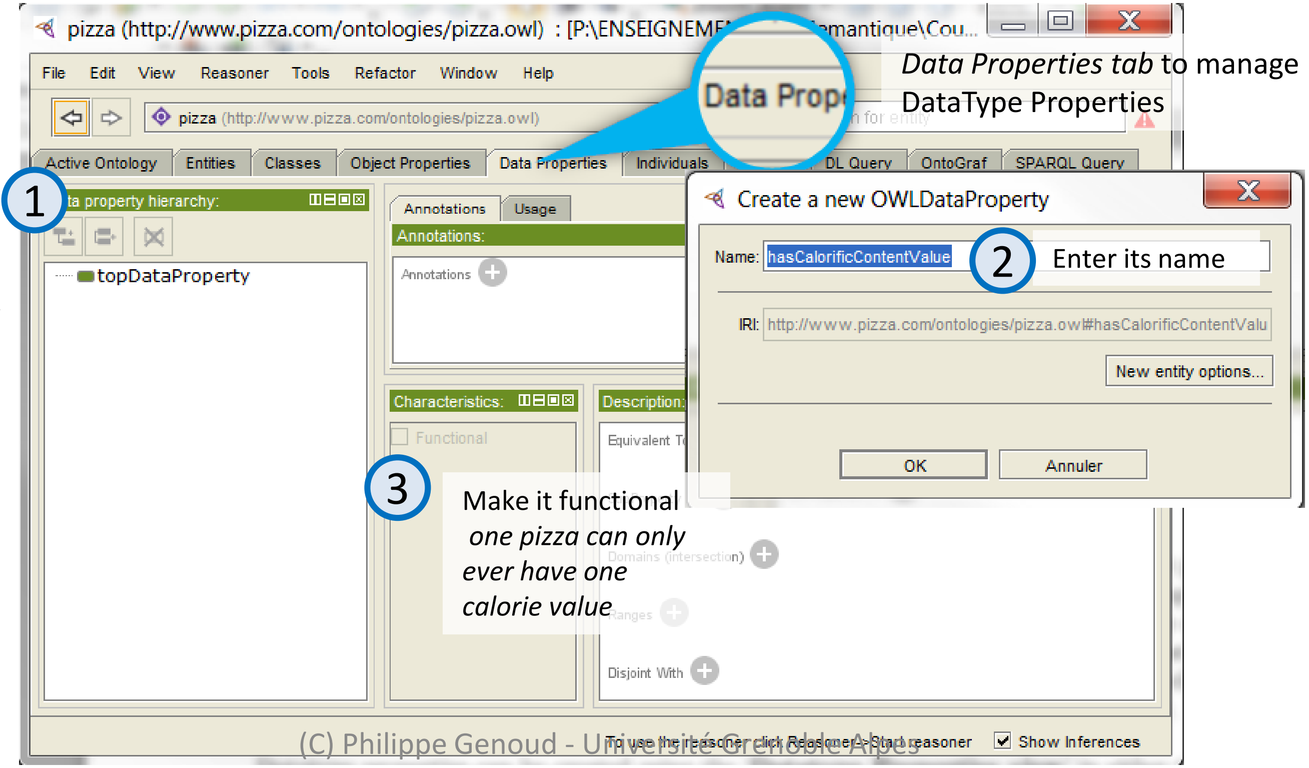
hasBase some PizzaBase

class description after step 5

DataType properties

- **DataType Property** : used to relate an individual to a concrete data value that may be typed (XML Schema Datatype) or untyped (rdf literal)

example: use some numeric ranges to broadly classify particular pizzas as high or low calorie.
→ a datatype property **hasCalorificContentValue** to state the calorie content of particular pizzas



Name: hasCalorificContentValue

IRI: http://www.pizza.com/ontologies/pizza.owl#hasCalorificContentValu

Data Properties tab to manage DataType Properties

Create a new DataType Property in the *Data property hierarchy*

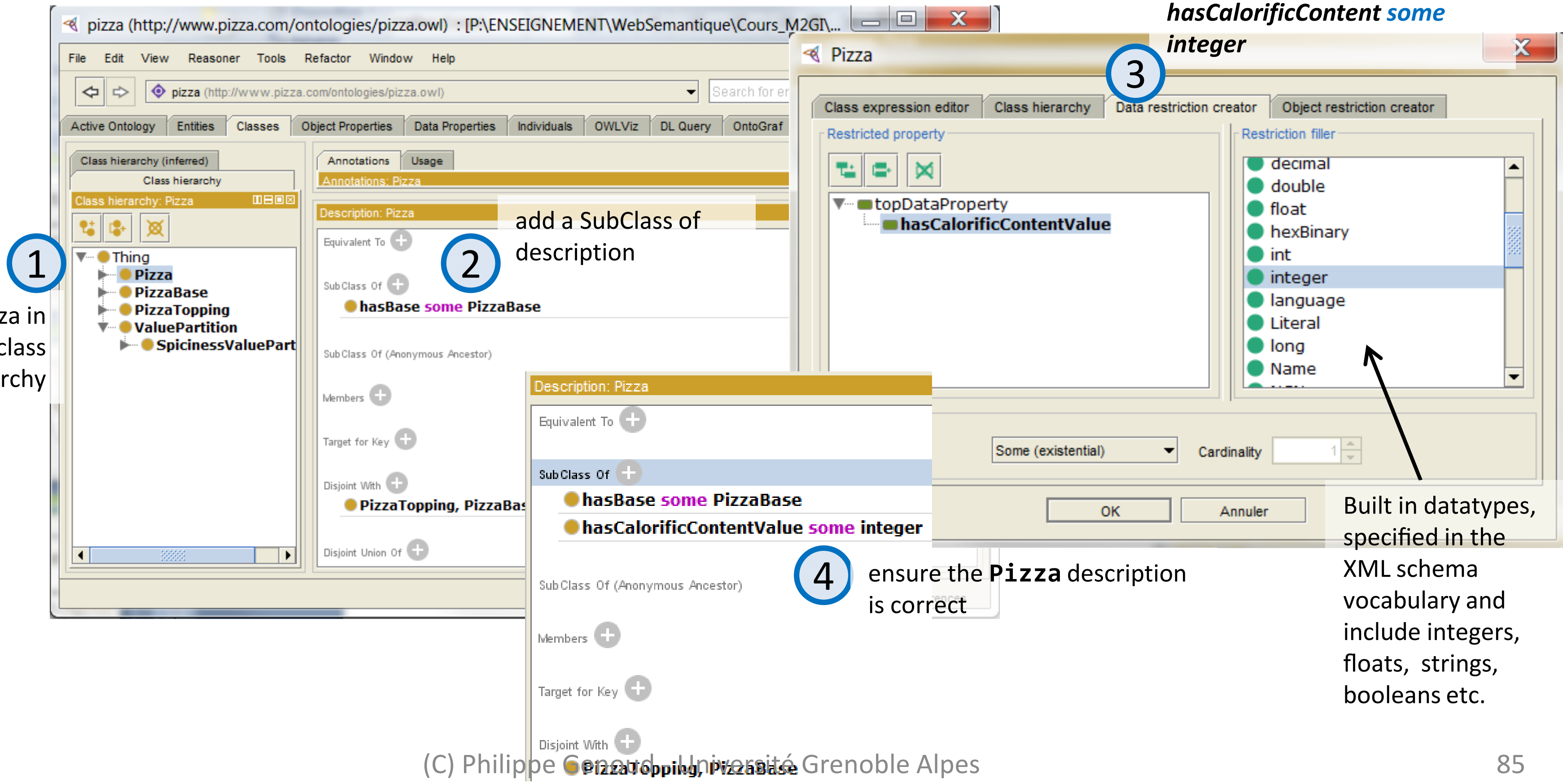
Enter its name

Make it functional
one pizza can only ever have one calorie value

using a DataType Property in a restriction

- A datatype property can also be used in a restriction to relate individuals to members of a given datatype.

Create a datatype restriction to state that all **Pizzas** have a calorific value



pizza (http://www.pizza.com/ontologies/pizza.owl) : [P:\ENSEIGNEMENT\WebSemantique\Cours_M2GI\...

Class hierarchy:
- Thing
  - Pizza
  - PizzaBase
  - PizzaTopping
  - ValuePartition
    - SpicinessValuePart

Restricted property:
- topDataProperty
  - hasCalorificContentValue

Restriction filler: decimal, double, float, hexBinary, int, integer, language, Literal, long, Name

Description: Pizza

SubClass Of
- hasBase some PizzaBase
- hasCalorificContentValue some integer

Disjoint With
- PizzaTopping, PizzaBase

1 Select Pizza in the class hierarchy

2 add a SubClass of description

3 In the *Data restriction creator* tab enter the restriction ***hasCalorificContent some integer***

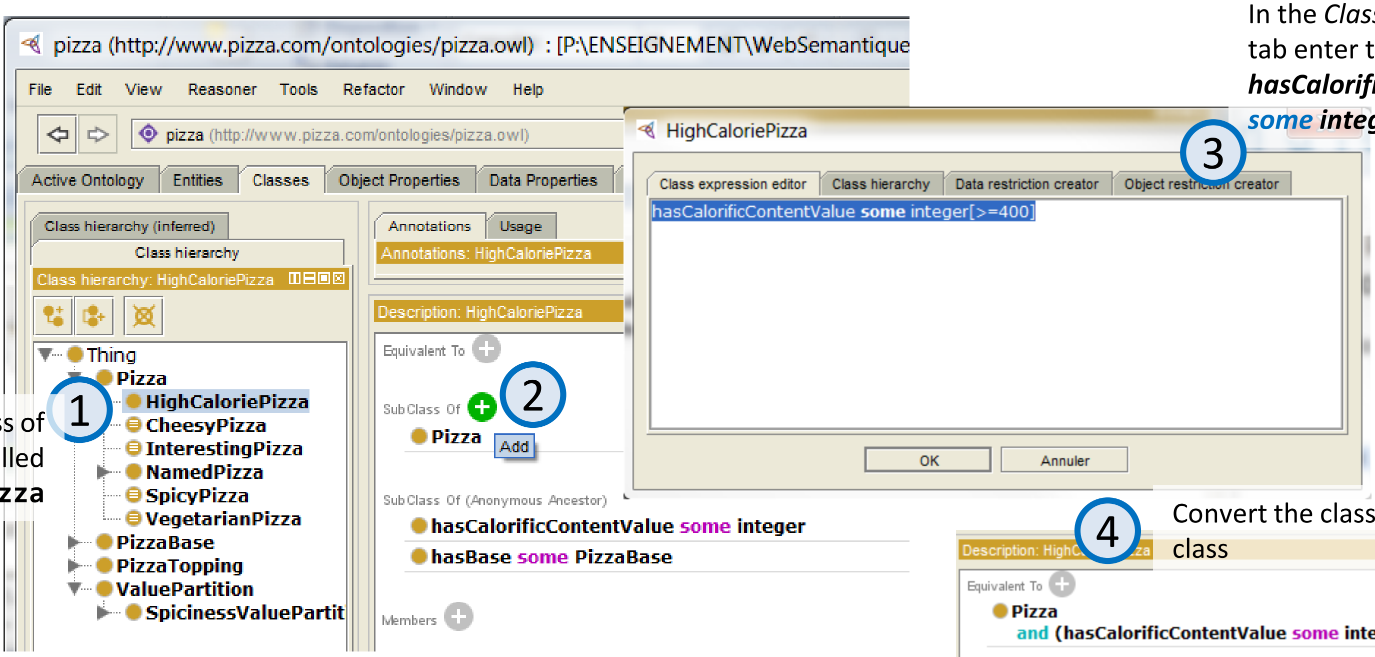
4 ensure the **Pizza** description is correct

Built in datatypes, specified in the XML schema vocabulary and include integers, floats, strings, booleans etc.

using a DataType Property in a restriction

- In addition to using the predefined set of datatypes it is possible to specialise the use of a datatype by specifying restrictions on the possible values..

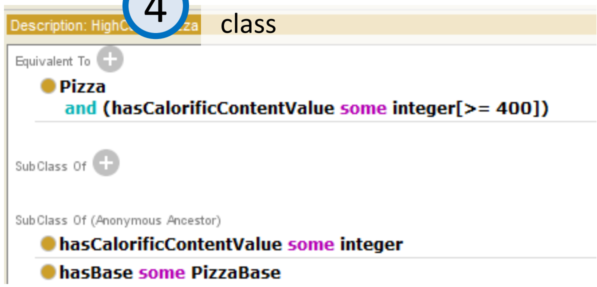
Create a **HighCaloriePizza** that has a calorific value higher than or equal to 400


pizza (http://www.pizza.com/ontologies/pizza.owl) : [P:\ENSEIGNEMENT\WebSemantique
File Edit View Reasoner Tools Refactor Window Help
pizza (http://www.pizza.com/ontologies/pizza.owl)
Active Ontology Entities Classes Object Properties Data Properties
Class hierarchy (inferred)
Class hierarchy
Class hierarchy: HighCaloriePizza
Thing
Pizza
HighCaloriePizza
CheesyPizza
InterestingPizza
NamedPizza
SpicyPizza
VegetarianPizza
PizzaBase
PizzaTopping
ValuePartition
SpicinessValuePartit
Annotations Usage
Annotations: HighCaloriePizza
Description: HighCaloriePizza
Equivalent To
SubClass Of
Pizza
Add
SubClass Of (Anonymous Ancestor)
hasCalorificContentValue some integer
hasBase some PizzaBase
Members
HighCaloriePizza
Class expression editor Class hierarchy Data restriction creator Object restriction creator
hasCalorificContentValue some integer[>=400]
OK Annuler
1
2
3

Create a subclass of **Pizza** called **HighCaloriePizza**

In the *Class expression editor* tab enter the restriction ***hasCalorificContentValue some integer[>=400]***

XSD minInclusive facet

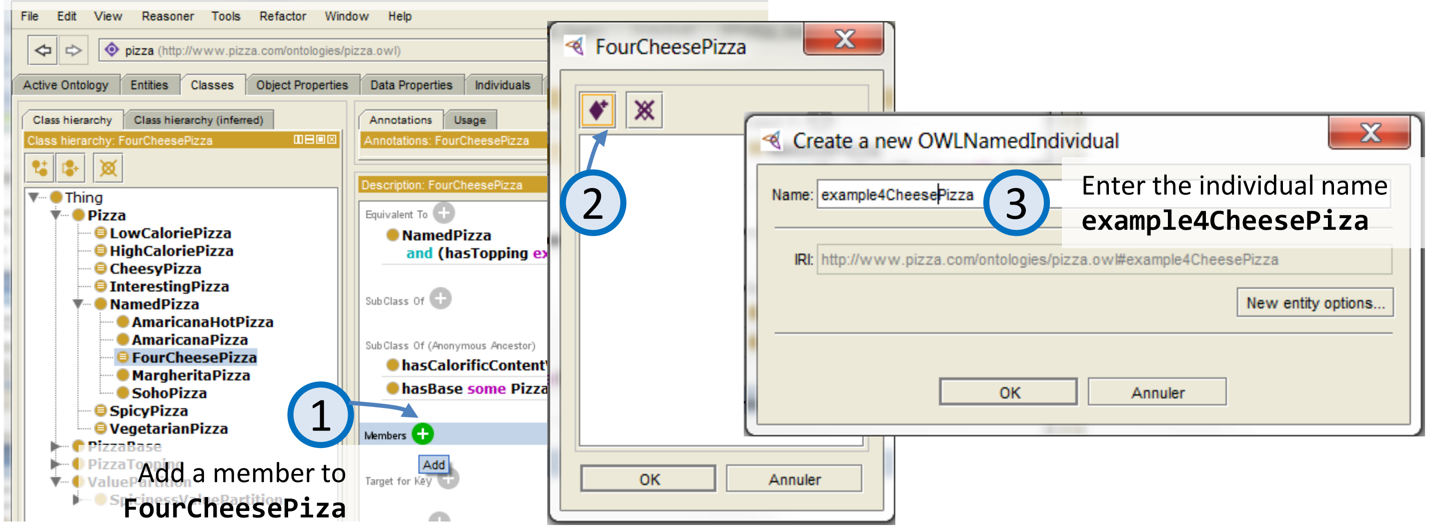

4
Description: HighC...za
Equivalent To
Pizza
and (hasCalorificContentValue some integer[>= 400])
SubClass Of
SubClass Of (Anonymous Ancestor)
hasCalorificContentValue some integer
hasBase some PizzaBase

Convert the class to a defined class

5 Create a **LowCaloriePizza** in the same way, but define it as being equivalent to **Pizza and (hasCalorificContentValue some integer[< 400])**

Creating individuals with DataType properties

Create an instance of **FourCheesePizza** with 723 calories



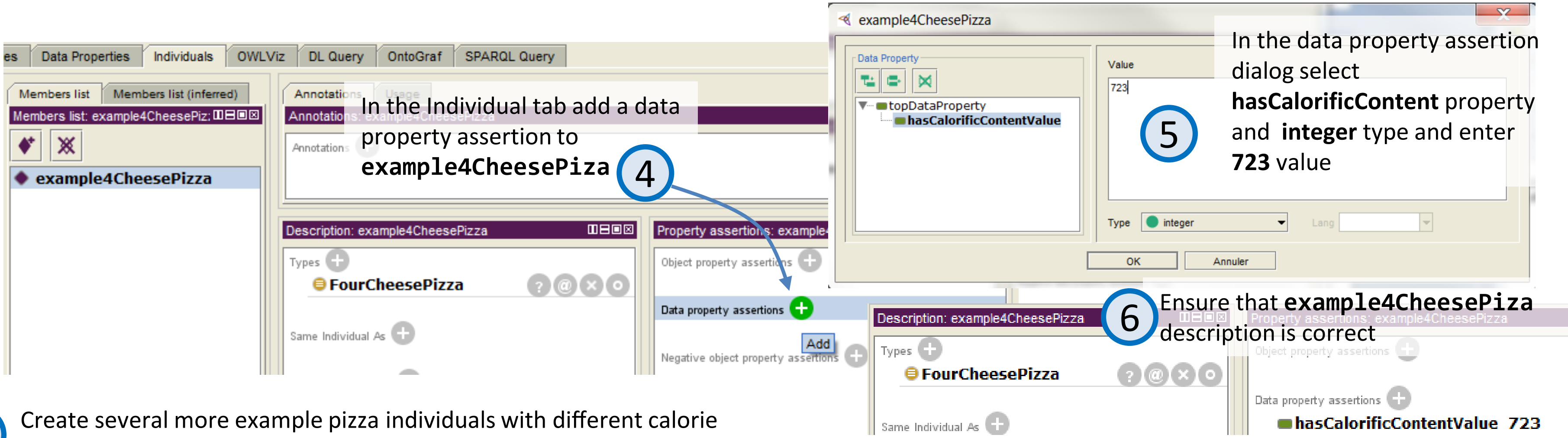
Name: example4CheesePizza

IRI: http://www.pizza.com/ontologies/pizza.owl#example4CheesePizza

1 Add a member to **FourCheesePiza**

2

3 Enter the individual name **example4CheesePiza**


es Data Properties Individuals OWLViz DL Query OntoGraf SPARQL Query
Members list Members list (inferred)
Members list: example4CheesePiz:
example4CheesePizza
Annotations Usage
Annotations: example4CheesePizza
Annotations
Description: example4CheesePizza
Types
FourCheesePizza
Same Individual As
Property assertions: example4
Object property assertions
Data property assertions
Add
Negative object property assertions
example4CheesePizza
Data Property
topDataProperty
hasCalorificContentValue
Value
Lang
OK
Annuler
Description: example4CheesePizza
Types
FourCheesePizza
Same Individual As
Property assertions: example4CheesePizza
Object property assertions
Data property assertions

Value: 723

Type: integer

hasCalorificContentValue 723

4 In the Individual tab add a data property assertion to **example4CheesePizza**

5 In the data property assertion dialog select **hasCalorificContent** property and **integer** type and enter **723** value

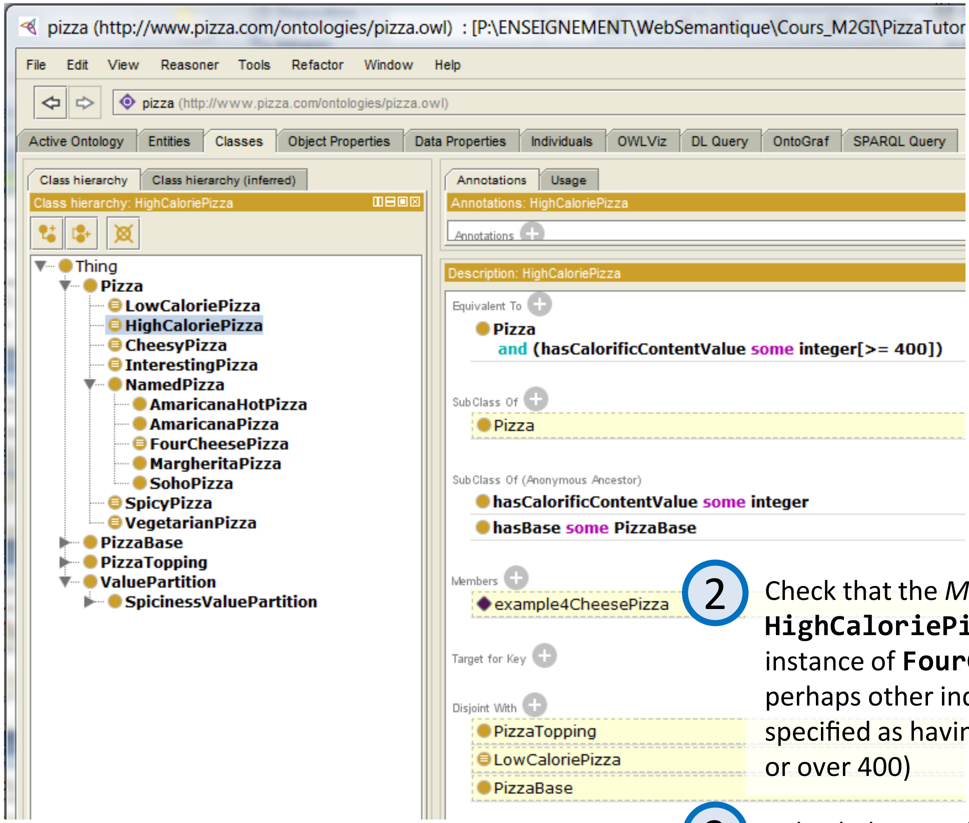
6 Ensure that **example4CheesePiza** description is correct

7 Create several more example pizza individuals with different calorie contents including an instance of **MargheritaPizza** with 263 calories

Performing instance classification

Classify pizza individuals based on their **hasCalorificContentValue**

1 Run a reasoner



pizza (http://www.pizza.com/ontologies/pizza.owl) : [P:\ENSEIGNEMENT\WebSemantique\Cours_M2GI\PizzaTutor

Class hierarchy: HighCaloriePizza

- Thing
  - Pizza
    - LowCaloriePizza
    - HighCaloriePizza
    - CheesyPizza
    - InterestingPizza
    - NamedPizza
      - AmericanaHotPizza
      - AmericanaPizza
      - FourCheesePizza
      - MargheritaPizza
      - SohoPizza
    - SpicyPizza
    - VegetarianPizza
  - PizzaBase
  - PizzaTopping
  - ValuePartition
    - SpicinessValuePartition

Annotations: HighCaloriePizza

Annotations

Description: HighCaloriePizza

Equivalent To

Pizza
and (hasCalorificContentValue some integer[>= 400])

SubClass Of

Pizza

SubClass Of (Anonymous Ancestor)

hasCalorificContentValue some integer

hasBase some PizzaBase

Members

example4CheesePizza

Target for Key

Disjoint With

PizzaTopping

LowCaloriePizza

PizzaBase

2 Check that the *Members* section of **HighCaloriePizza** contains your instance of **FourCheesePizza** (and perhaps other individuals which you specified as having a calorie value equal to or over 400)

3 Check the members of **LowCalorie** Pizza

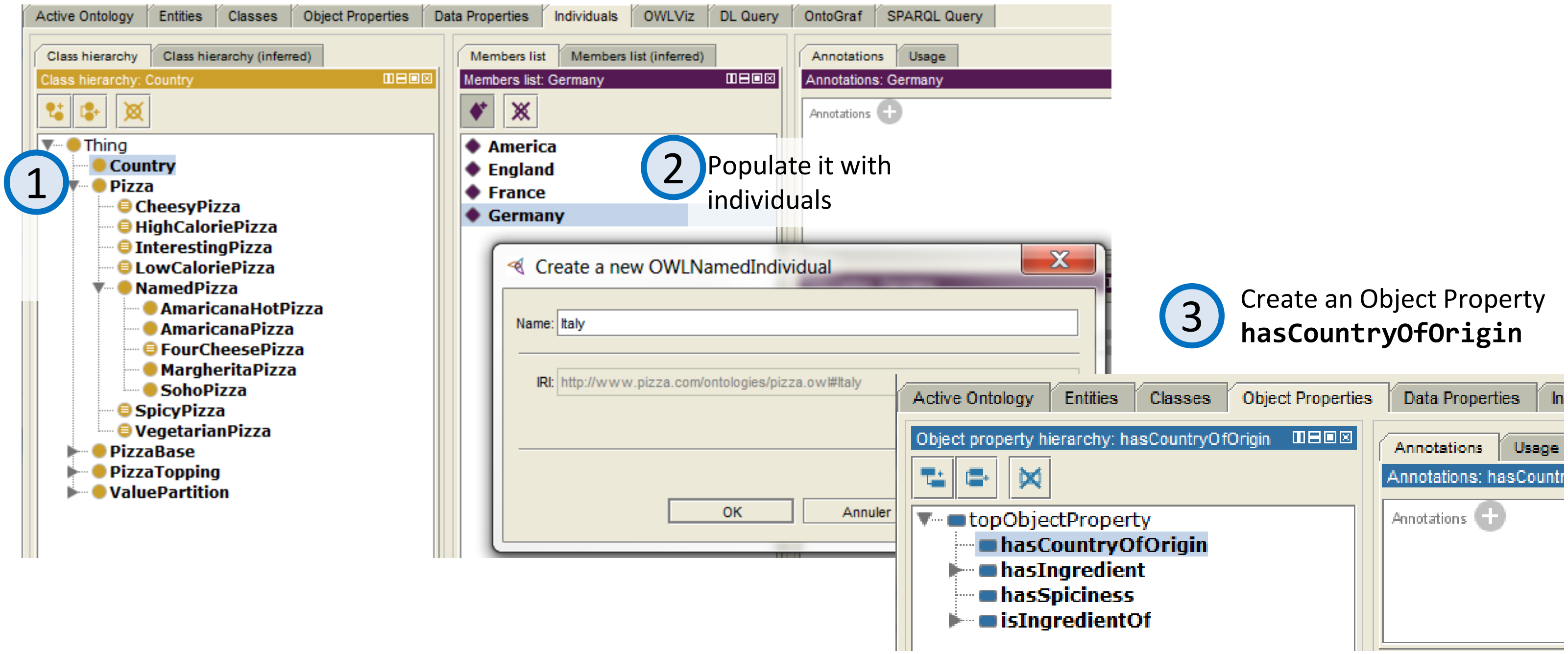
There is a bug in Protégé 4.3. , inferred Members do not appear immediately on the class description view.

You might need to turn on inferences for individuals. In the preferences select the "Reasoner" tab. Look at the section "Displayed Individual Inferences" and check the various boxes an necessary.

You can also use the DL query tab. Type "HighCaloriePizza" into the query editor and make sure "Instances" is selected on the right hand side.

hasValue Restrictions

- **hasValue Restriction**
  - describes the set of individuals that have at least one relationship along a specified property to a specific individual.
  - example : to describe the country of origin of various pizza toppings



Create a new OWLNamedIndividual

Name: Italy

IRI: http://www.pizza.com/ontologies/pizza.owl#Italy

1 reate Country a ubclass of Thing

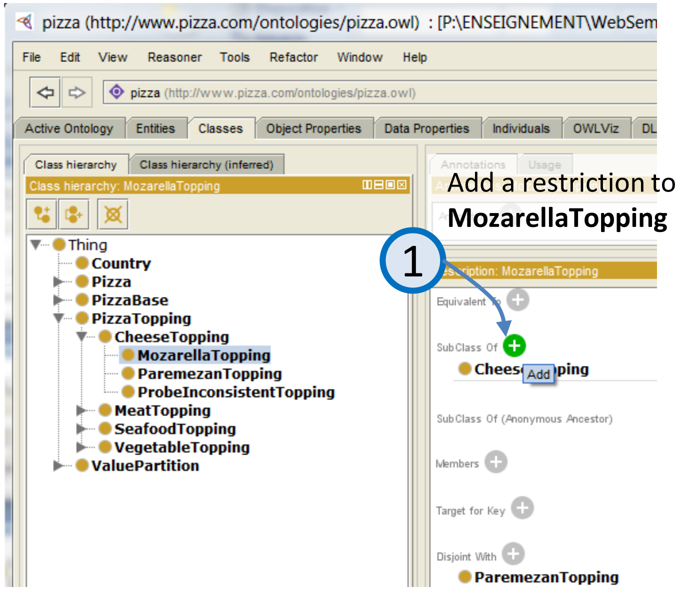
2 Populate it with individuals

3 Create an Object Property **hasCountryOfOrigin**

hasValue Restrictions

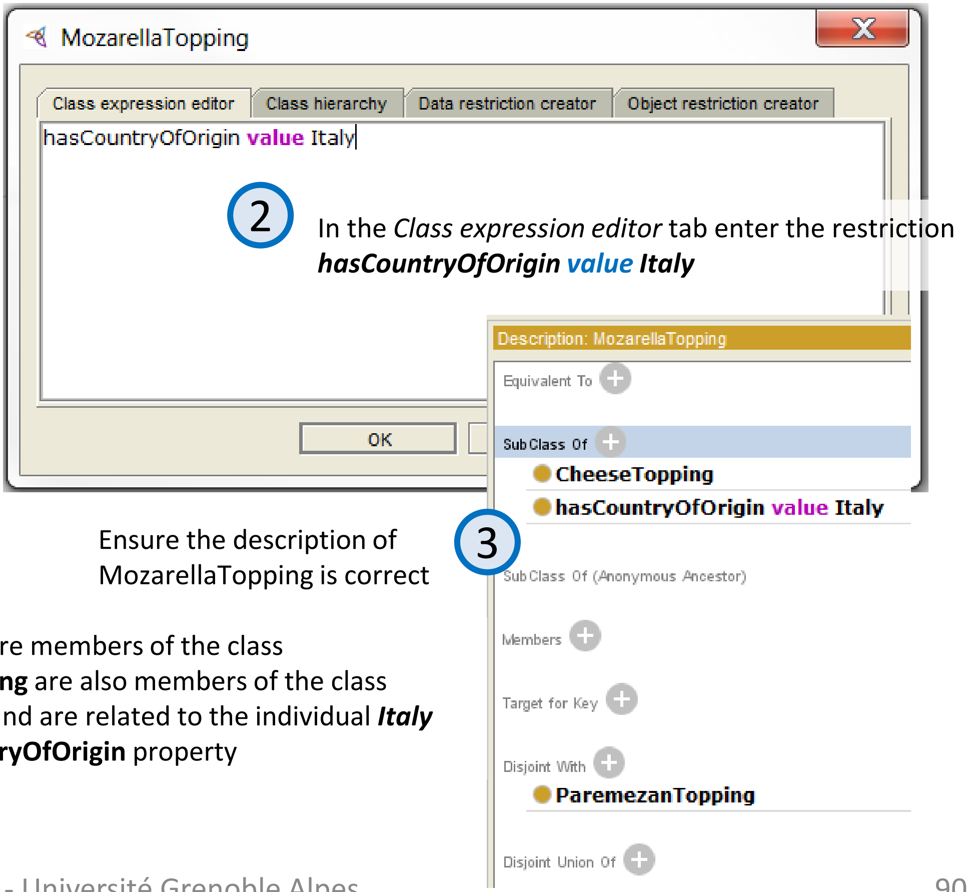
example : to describe the country of origin of various pizza toppings (continued)

Create a **hasValue** restriction to specify that **MozzarellaTopping** has Italy as its country of origin.



pizza (http://www.pizza.com/ontologies/pizza.owl) : [P:\ENSEIGNEMENT\WebSem
File Edit View Reasoner Tools Refactor Window Help
pizza (http://www.pizza.com/ontologies/pizza.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL
Class hierarchy Class hierarchy (inferred)
Class hierarchy: MozarellaTopping
Thing
Country
Pizza
PizzaBase
PizzaTopping
CheeseTopping
MozarellaTopping
ParemezanTopping
ProbeInconsistentTopping
MeatTopping
SeafoodTopping
VegetableTopping
ValuePartition
Annotations Usage
Description: MozarellaTopping
Equivalent To
SubClass Of
CheeseTopping
Add
SubClass Of (Anonymous Ancestor)
Members
Target for Key
Disjoint With
ParemezanTopping

1 Add a restriction to **MozzarellaTopping**



MozarellaTopping
Class expression editor Class hierarchy Data restriction creator Object restriction creator
OK
Description: MozarellaTopping
Equivalent To
SubClass Of
CheeseTopping
SubClass Of (Anonymous Ancestor)
Members
Target for Key
Disjoint With
ParemezanTopping
Disjoint Union Of

hasCountryOfOrigin **value** Italy

2 In the *Class expression editor* tab enter the restriction ***hasCountryOfOrigin value Italy***

3 Ensure the description of MozarellaTopping is correct

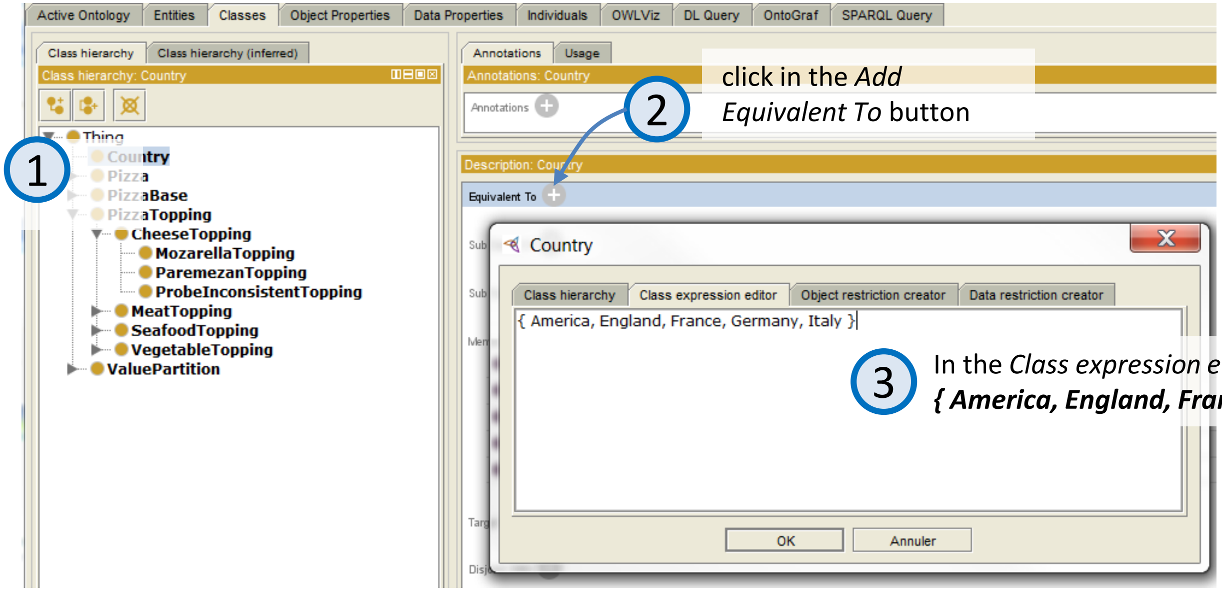
hasCountryOfOrigin value Italy

individuals that are members of the class **MozzarellaTopping** are also members of the class **CheeseTopping** and are related to the individual ***Italy*** via the **hasCountryOfOrigin** property

With current reasoners the classification is not complete for individuals. Use individuals in class descriptions with care — unexpected results may be caused by the reasoner.

Enumerated Classes

- **Enumerated class**
  - a class defined by precisely listing the individuals that are the members of it.
  - Enumerated classes described in this way are anonymous classes
    - they are the class of the individuals (and only the individuals) listed in the enumeration.
  - we can attach these individuals to a named class by creating the enumeration as an equivalent class.
  - example
    - Create an enumerated class four countries `{ America, England, France, Germany, Italy }`



{ America, England, France, Germany, Italy }

1 select **Country**

2 click in the *Add Equivalent To* button

3 In the *Class expression editor* tab enter the restriction ***{ America, England, France, Germany, Italy }***