Protégé 4: Building an OWL Ontology

May 2008

Mari Carmen Suárez-Figueroa, Oscar Corcho

{mcsuarez, ocorcho}@fi.upm.es
Ontology Engineering Group. Departamento de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid

Named Classes (I)

An ontology contains **classes** – indeed, the main building blocks of an OWL ontology are classes.

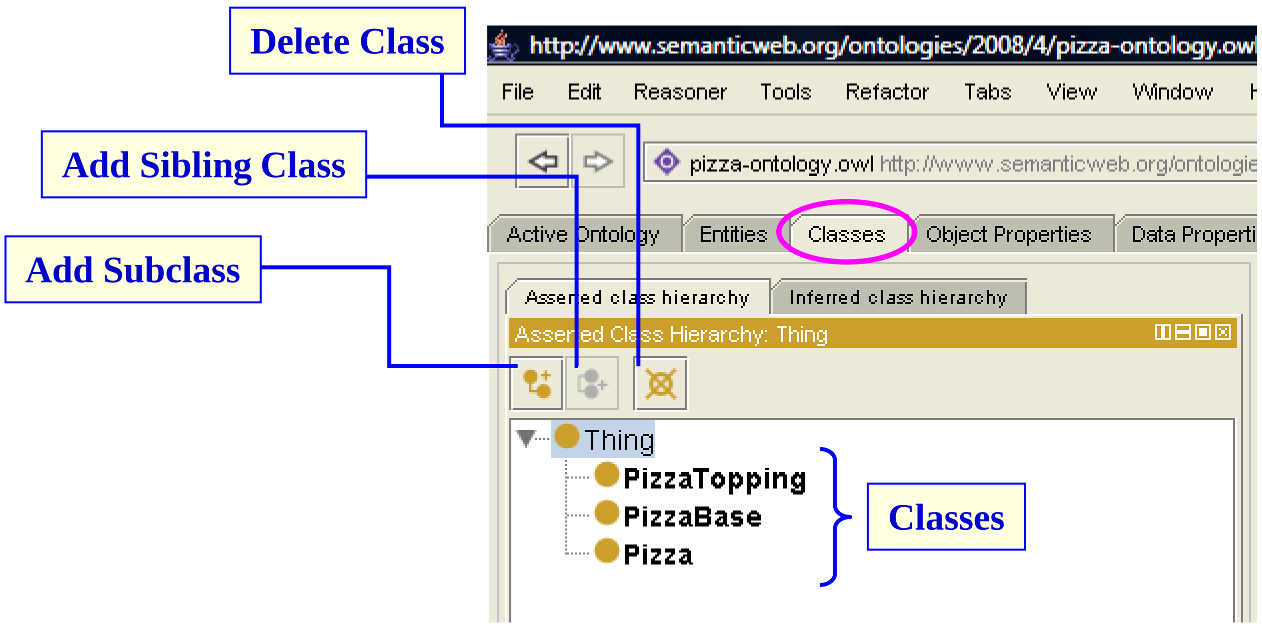
An empty ontology contains one class called *Thing*.

OWL classes are interpreted as sets of individuals or sets of objects. The class Thing is the class that represents the set containing all individuals.

Because of this all classes are subclasses of Thing.

Named Classes (II)

Creating classes in the pizza example: Pizza, PizzaBase, and PizzaTopping.



Disjoint Classes (I)

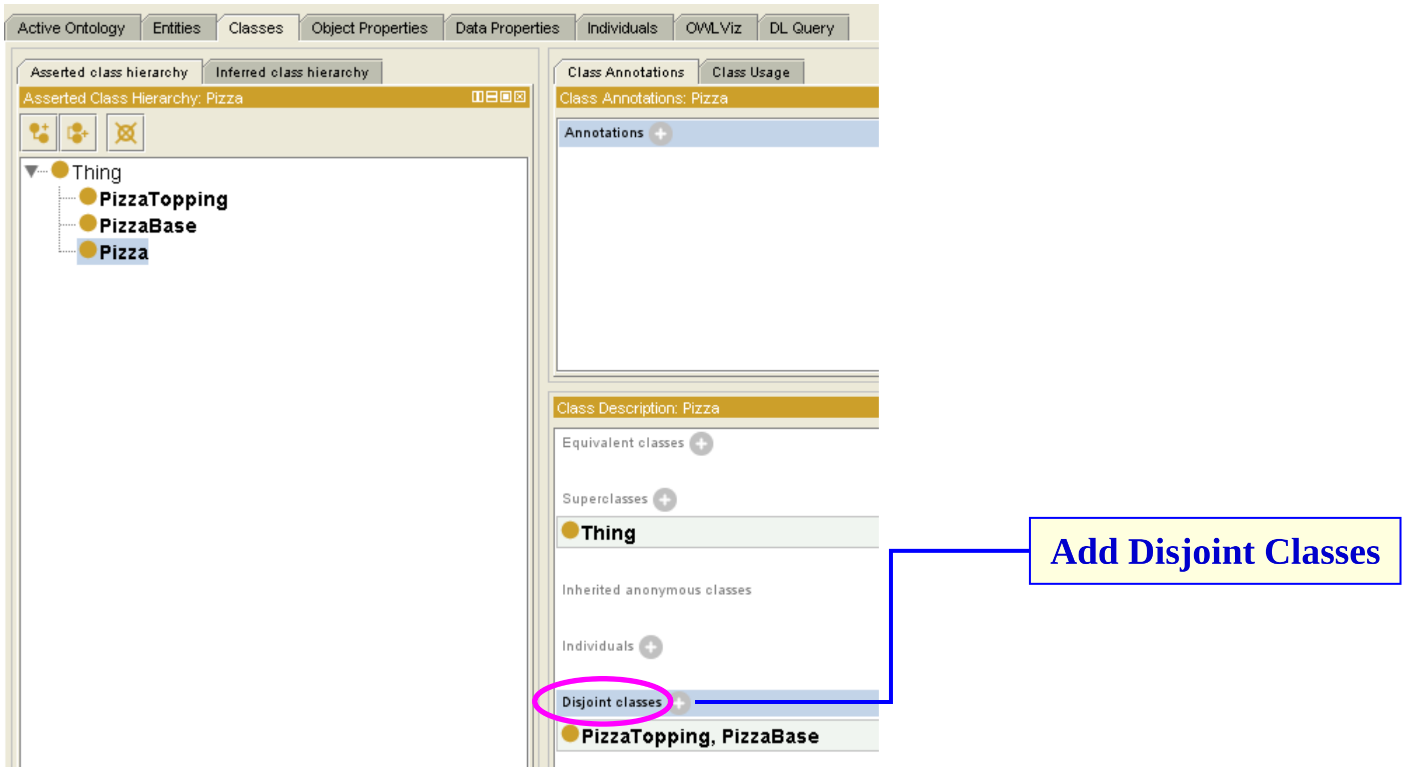
OWL Classes are assumed to ‘**overlap**’. We therefore cannot assume that an individual is not a member of a particular class simply because it has not been asserted to be a member of that class.

In order to ‘separate’ a group of classes we must make them **disjoint** from one another. This ensures that an individual which has been asserted to be a member of one of the classes in the group cannot be a member of any other classes in that group.

Disjoint Classes (II)

Making the classes Pizza, PizzaTopping and PizzaBase disjoint from one another.

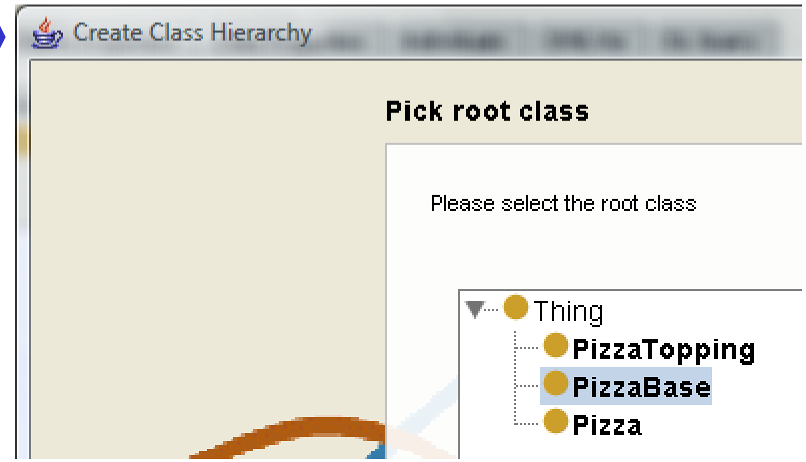
This means that it is not possible for an individual to be a member of a combination of these classes – it would not make sense for an individual to be a Pizza and a PizzaBase.



Named Classes (III)

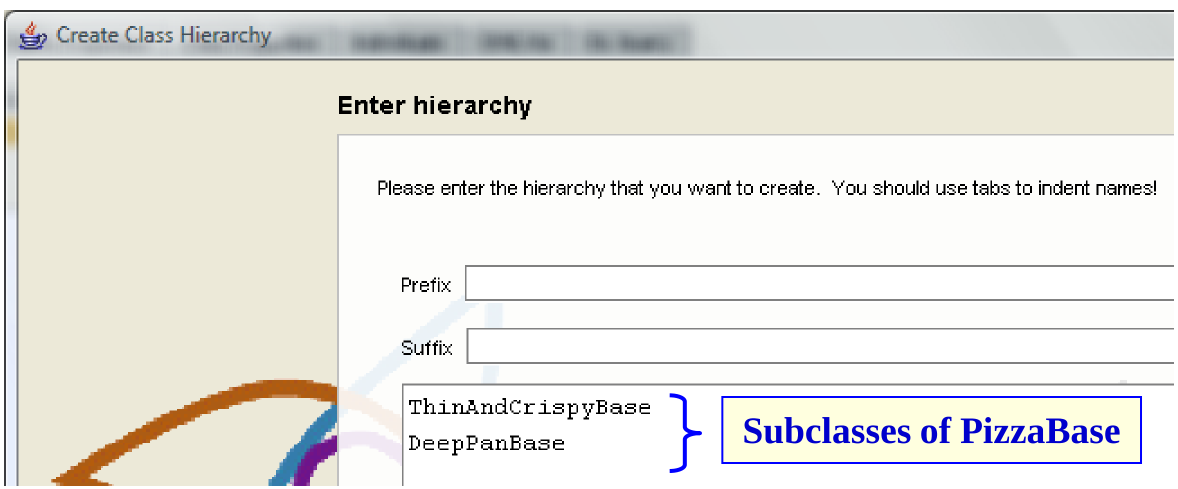
Creating subclasses in the pizza example: ThinAndCrispyBase and DeepPanBase.

2

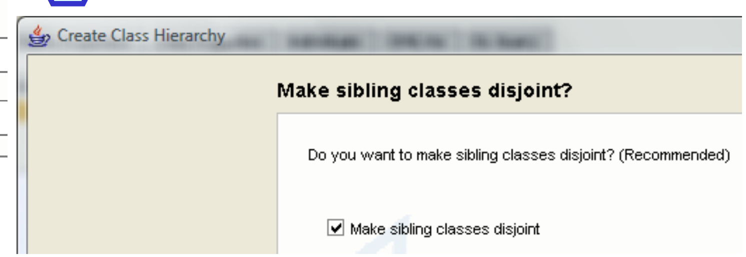


1

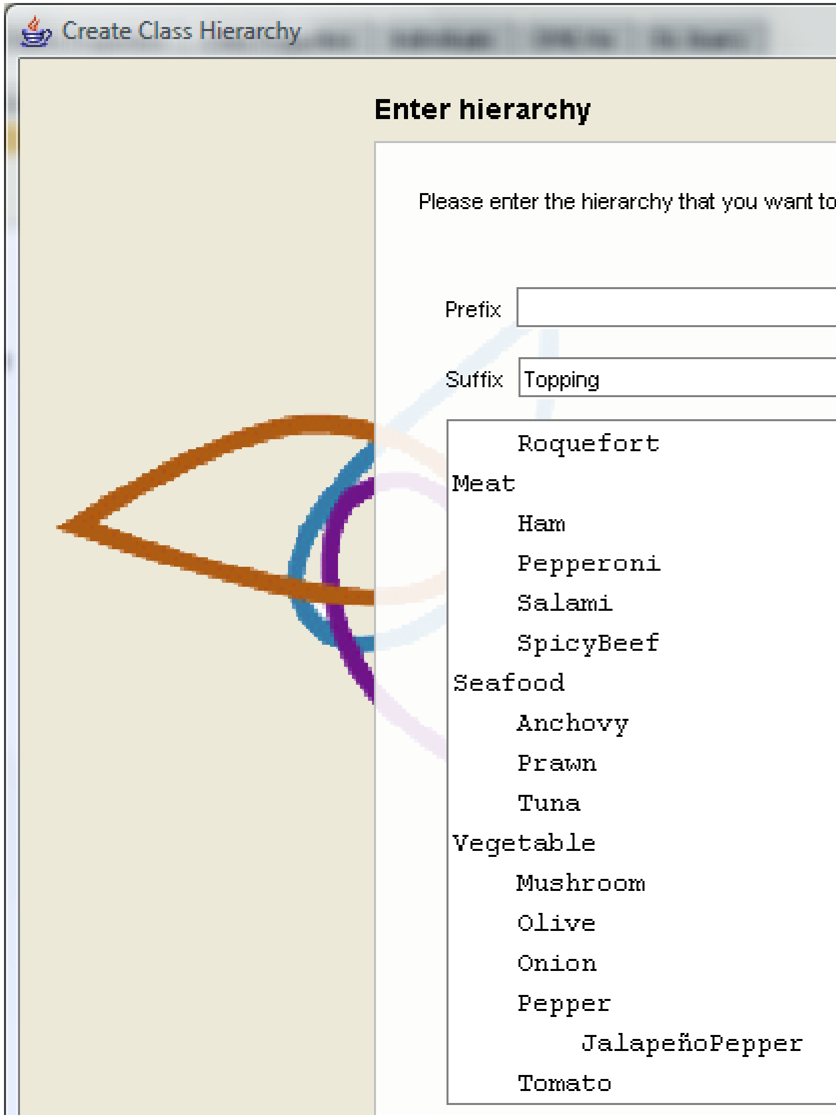
3

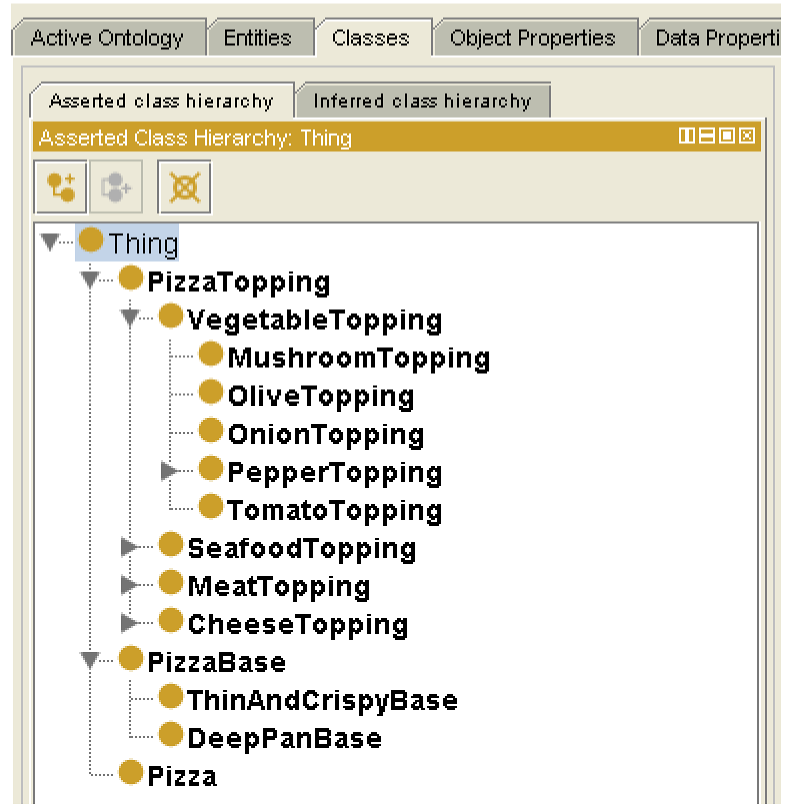


4



Class Hierarchy (I)

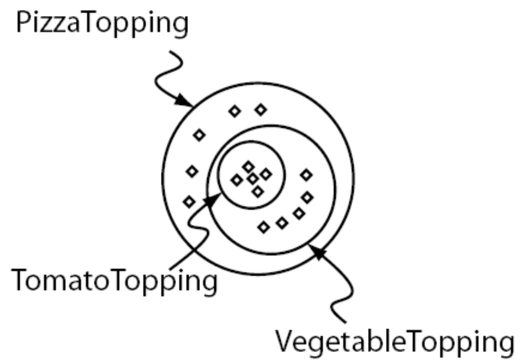




Class Hierarchy (II)

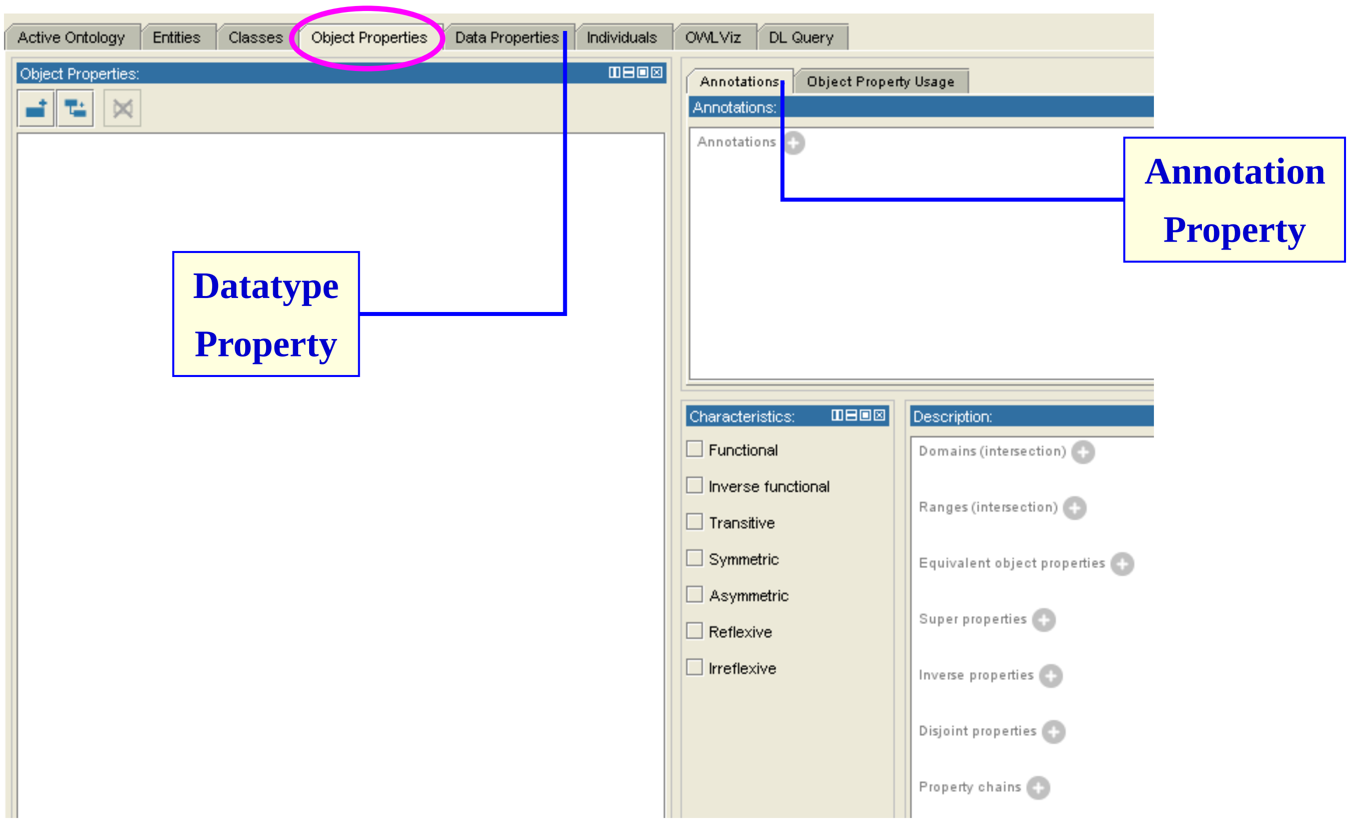
The Meaning of Subclass: all individuals that are members of the class TomatoTopping are members of the class VegetableTopping and members of the class PizzaTopping as we have stated that TomatoTopping is a subclass of VegetableTopping which is a subclass of PizzaTopping.

In OWL subclass means necessary implication. In other words, if VegetableTopping is a subclass of PizzaTopping then ALL instances of VegetableTopping are instances of PizzaTopping, without exception — if something is a VegetableTopping then this implies that it is also a PizzaTopping.



OWL Properties

OWL Properties represent relationships. There are two main types: Object properties and Datatype properties. OWL also has a third type of property: Annotation properties.



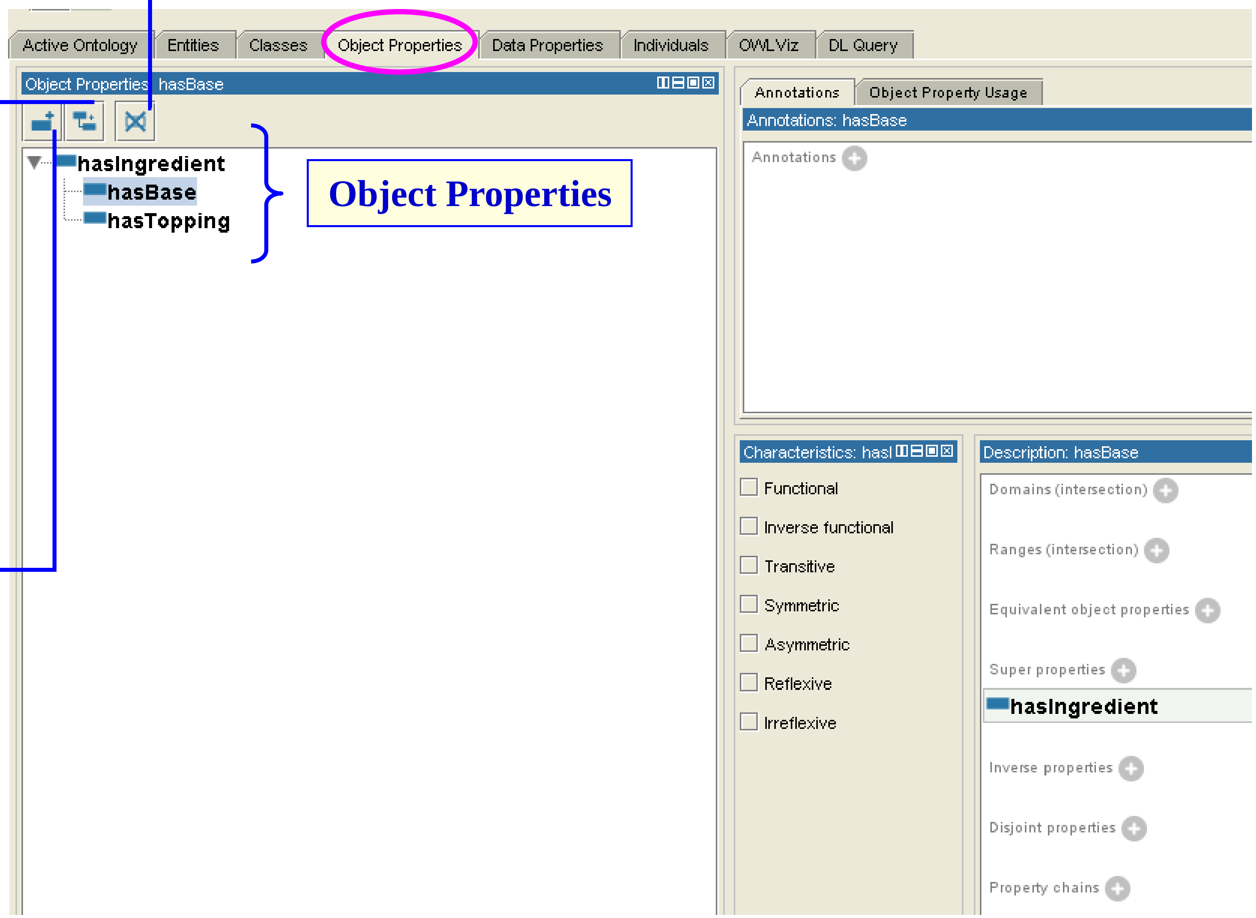
Object Properties

Creating object properties in the pizza example: hasIngredient, hasBase, and hasTopping.

Delete Object Property

Add Subproperty

Add Object Property

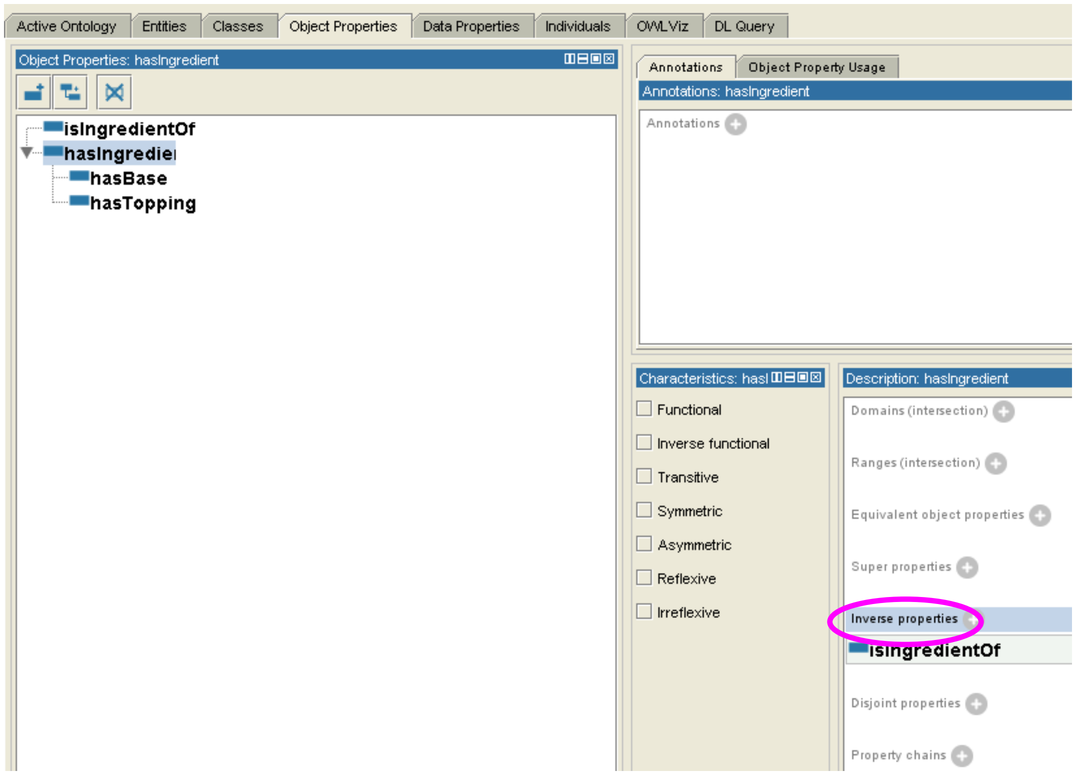


Inverse Properties

Each object property may have a corresponding inverse property.

If some property links individual a to individual b, then its invers property ling individual b to individual a.

Creating inverse properties in the pizza example: isIngredientOf.



OWL Object Properties Characteristics (I)

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

Functional Properties

Inverse Functional Properties

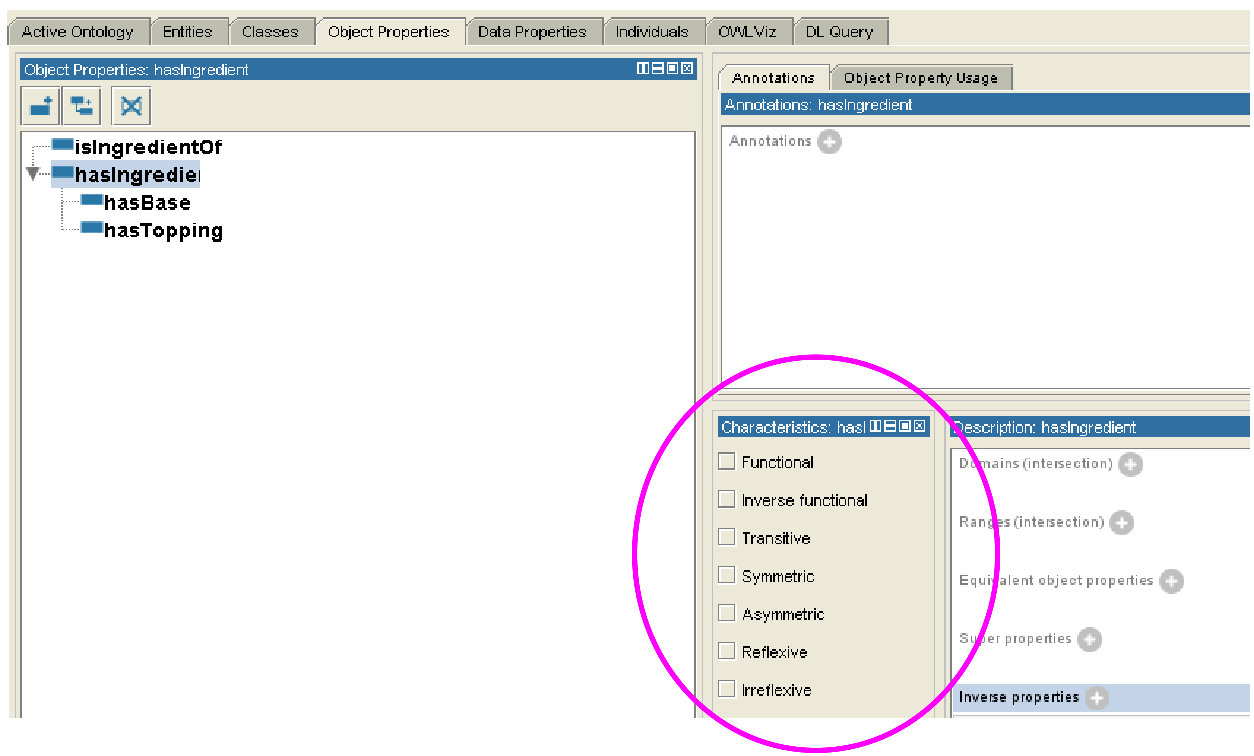
Transitive Properties

Symmetric Properties

Antisymetric Properties

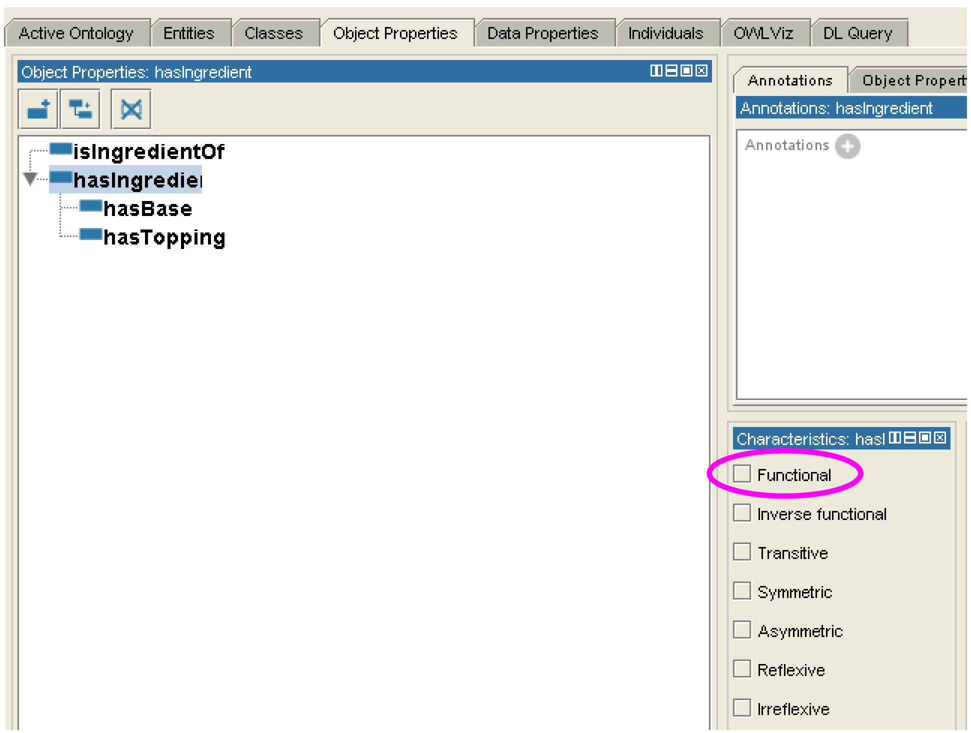
Reflexive Properties

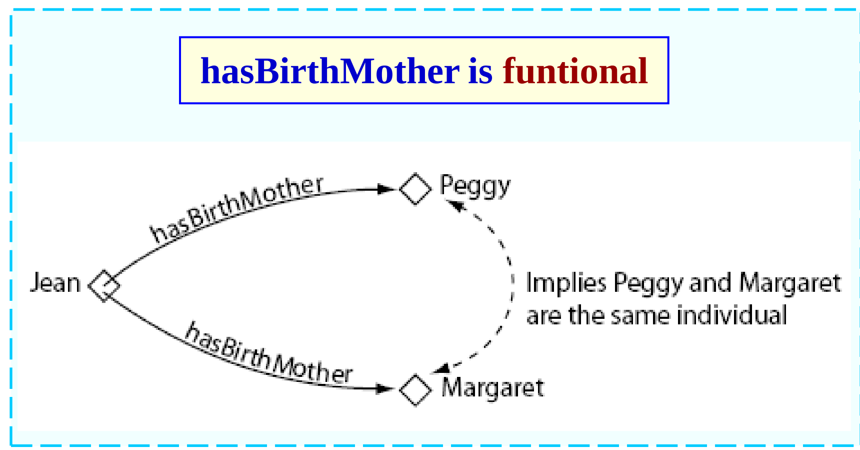
Irreflexive Properties



Functional Properties

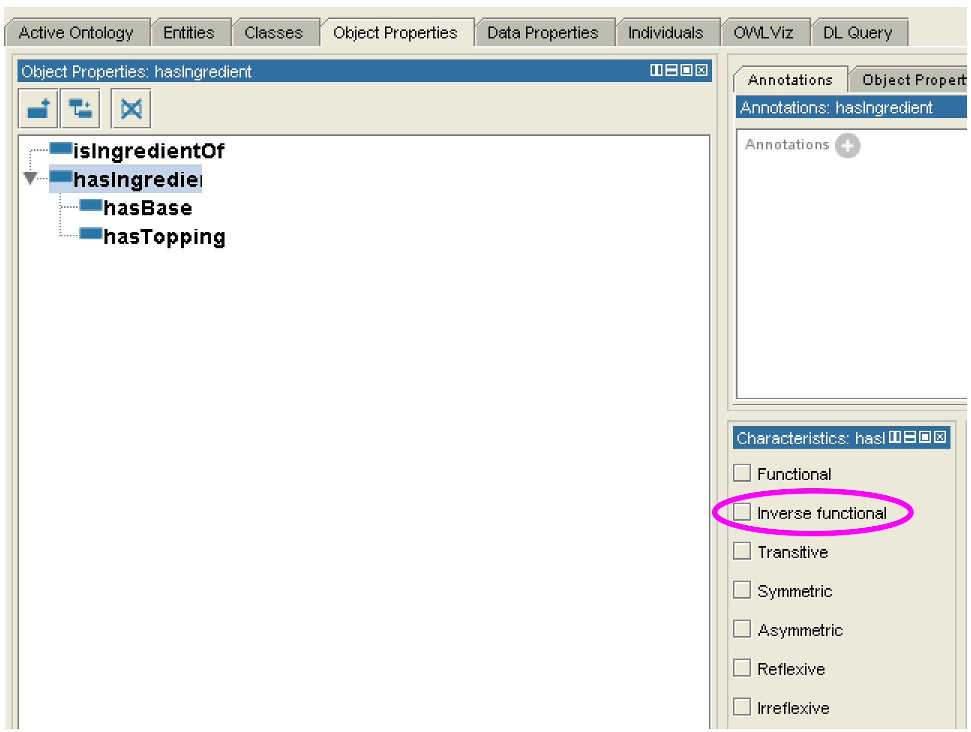
If a property is **functional**, for a given individual, there can be at most one individual that is related to the individual via the property.



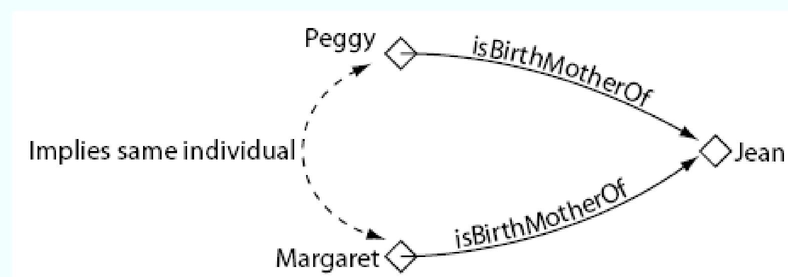


Inverse Functional Properties

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.

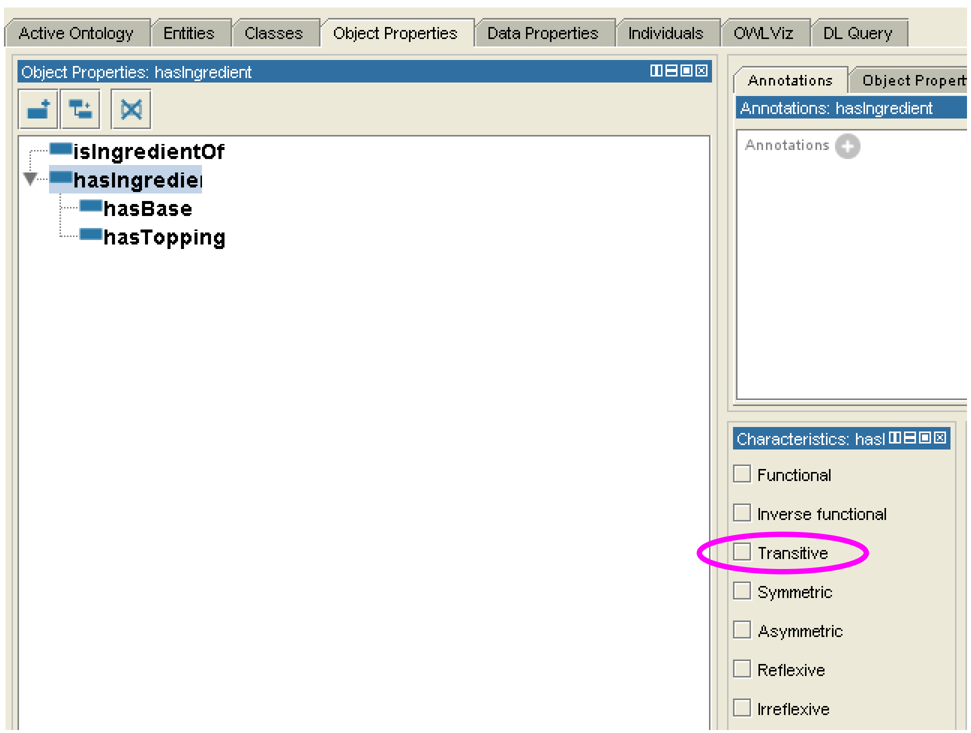


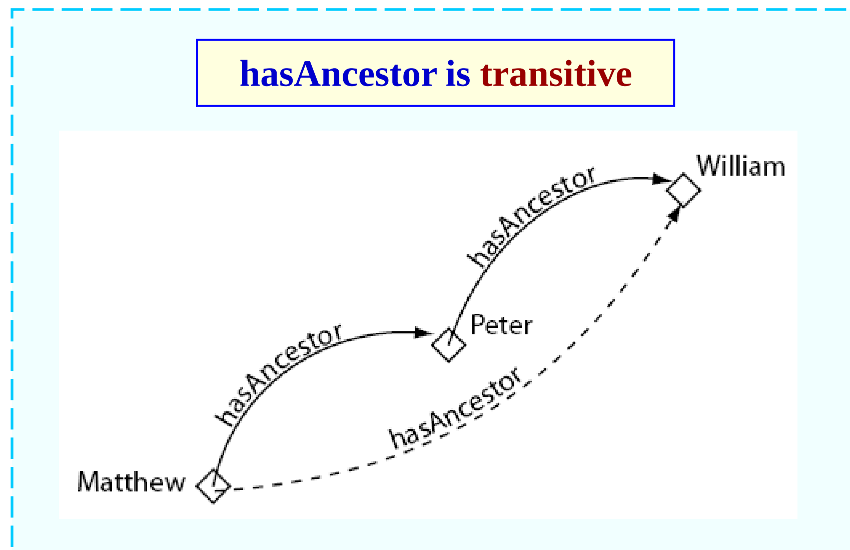
isBirthMotherOf is inverse functional
This is the inverse property of hasBirthMother — since hasBirthMother is functional, isBirthMotherOf is inverse functional



Transitive Properties

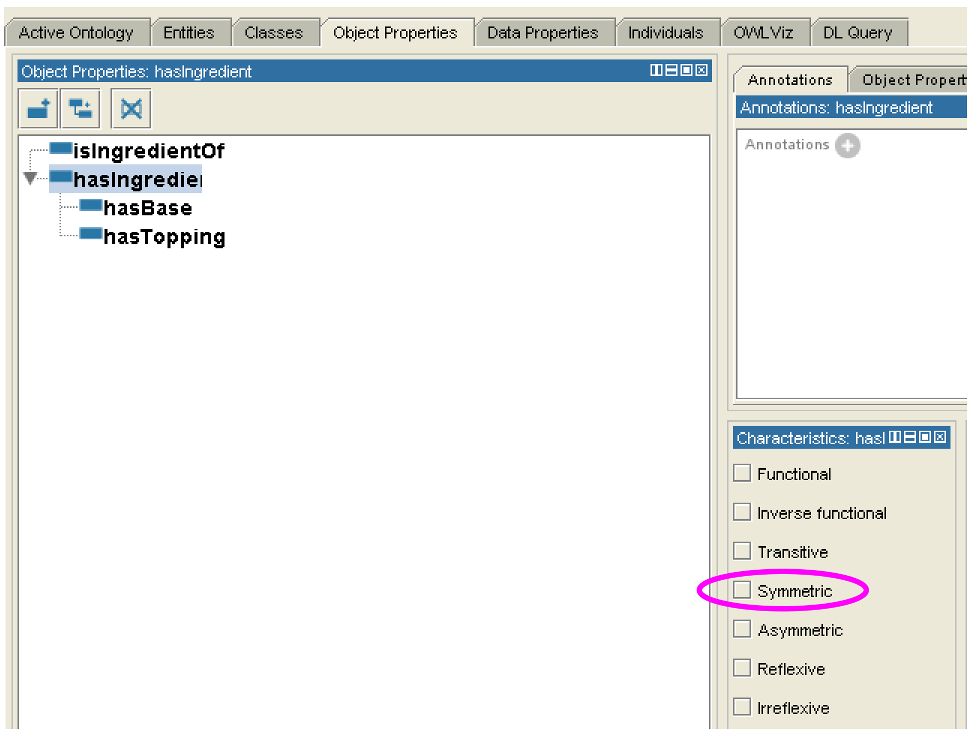
If a property is **transitive**, and the property relates individual a to individual b, and also individual b to individual c, then we can infer that individual a is related to individual c via the property.

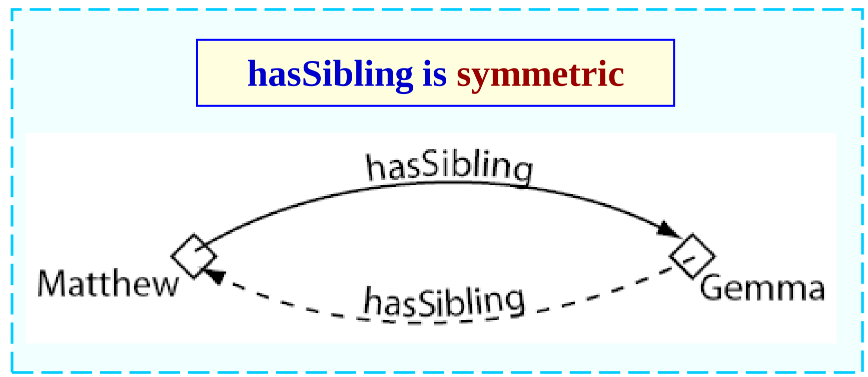




Symmetric Properties

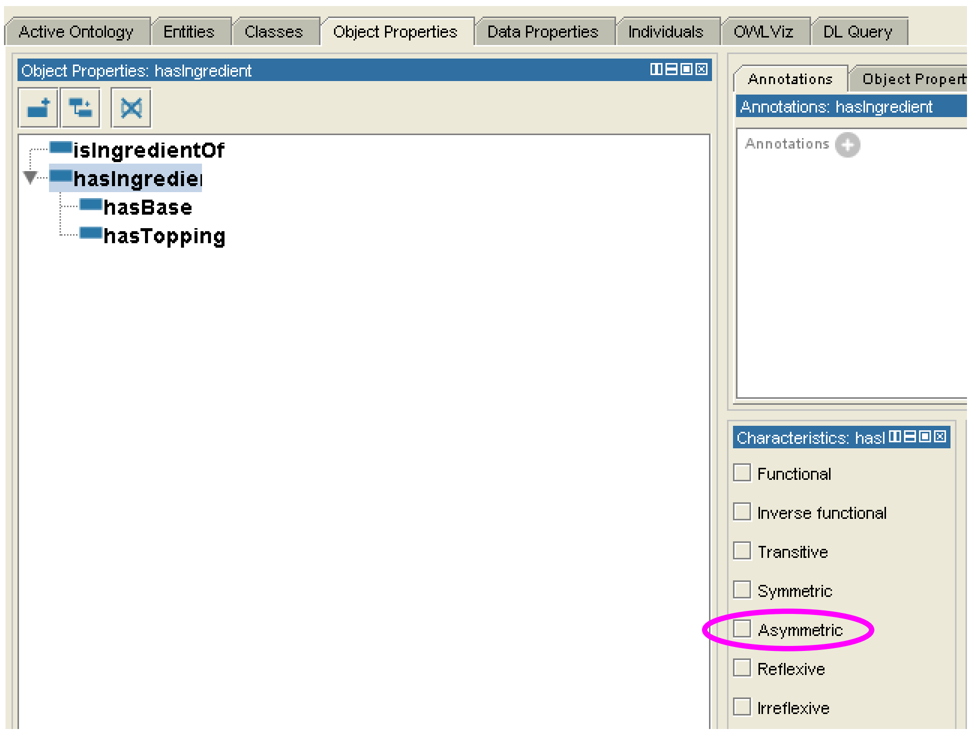
If a property is **symmetric**, and the property relates individual a to individual b then individual b is also related to individual a via the property.



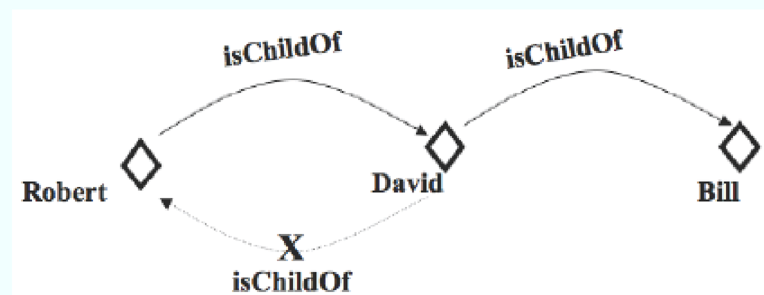


Antisymmetric Properties

If a property is **antisymmetric**, and the property relates individual a to individual b then individual b cannot be related to individual a via the property.

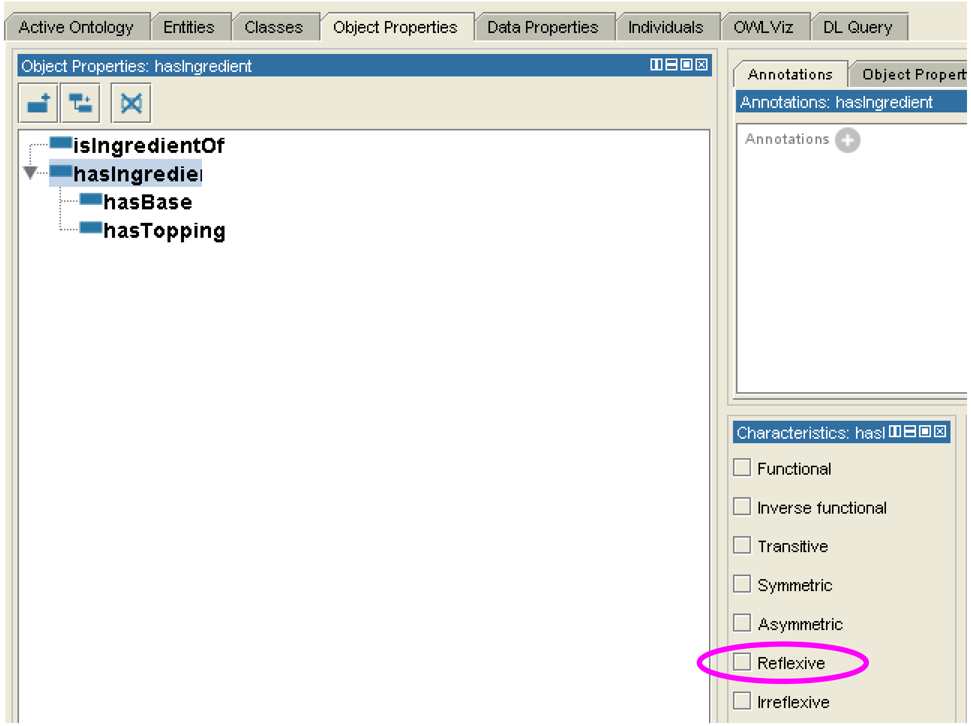


isChildOf is antisymmetric

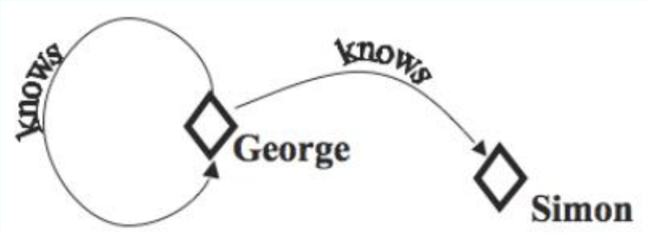


Reflexive Properties

A property is said to be **reflexive** when the property must relate individual a to itself.

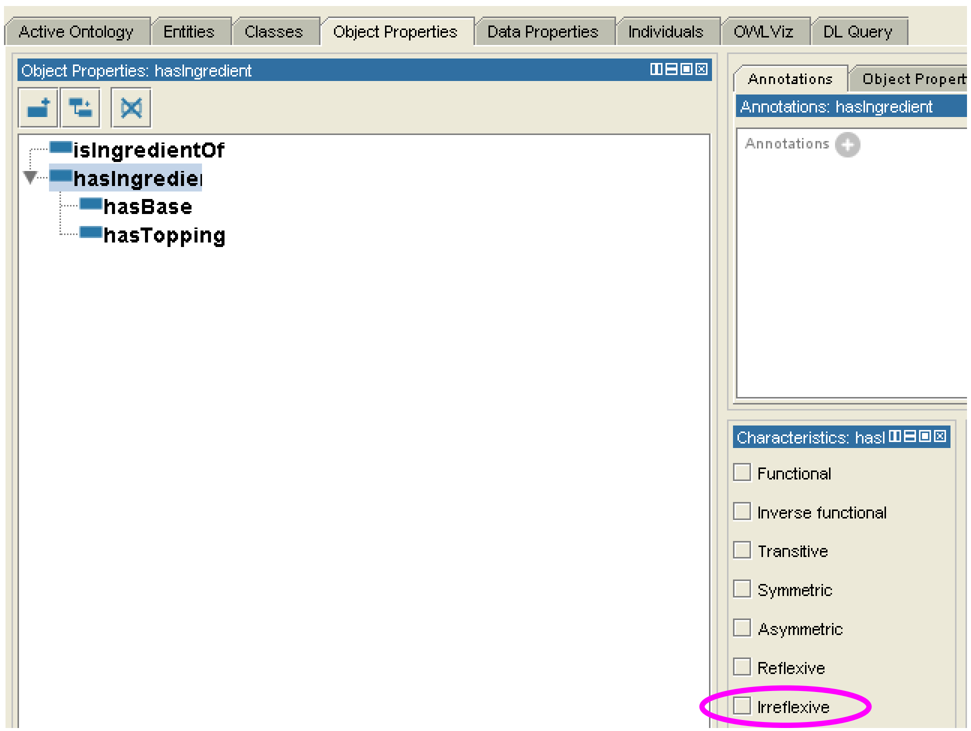


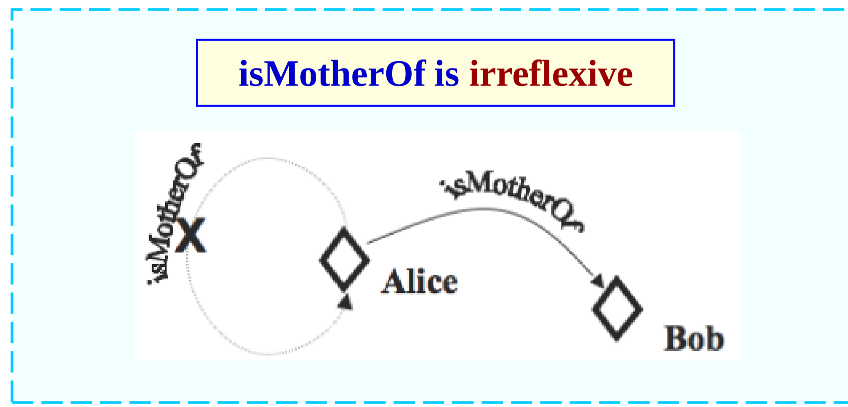
knows is reflexive



Irreflexive Properties

If a property 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.





Exercise

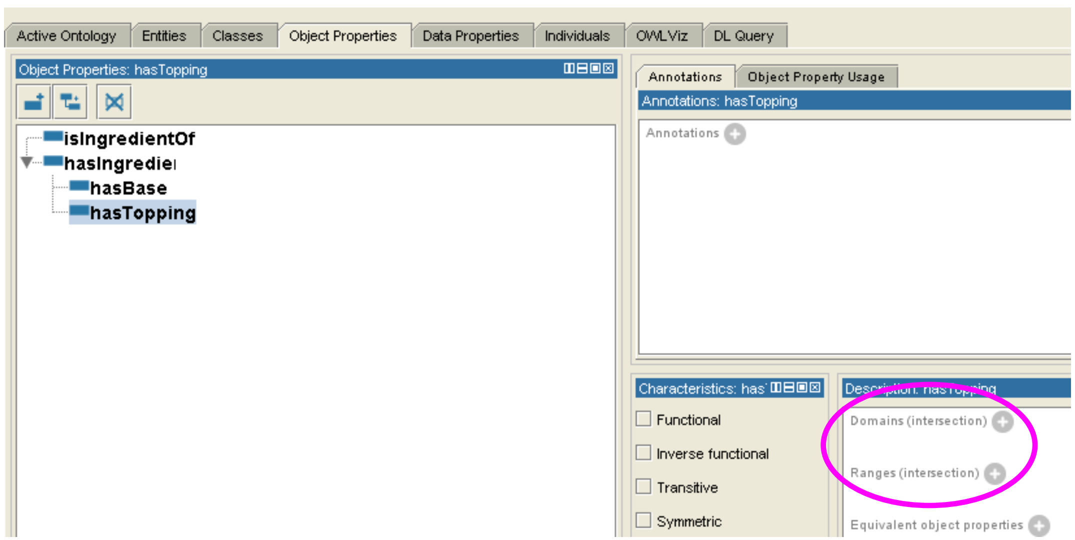
Modelling that a pizza has only one pizza base; and that if a pizza topping has ingredients, then the pizza itself contains also such ingredients.

OWL Properties: Domain and Range (I)

Properties may have a domain and a range specified.

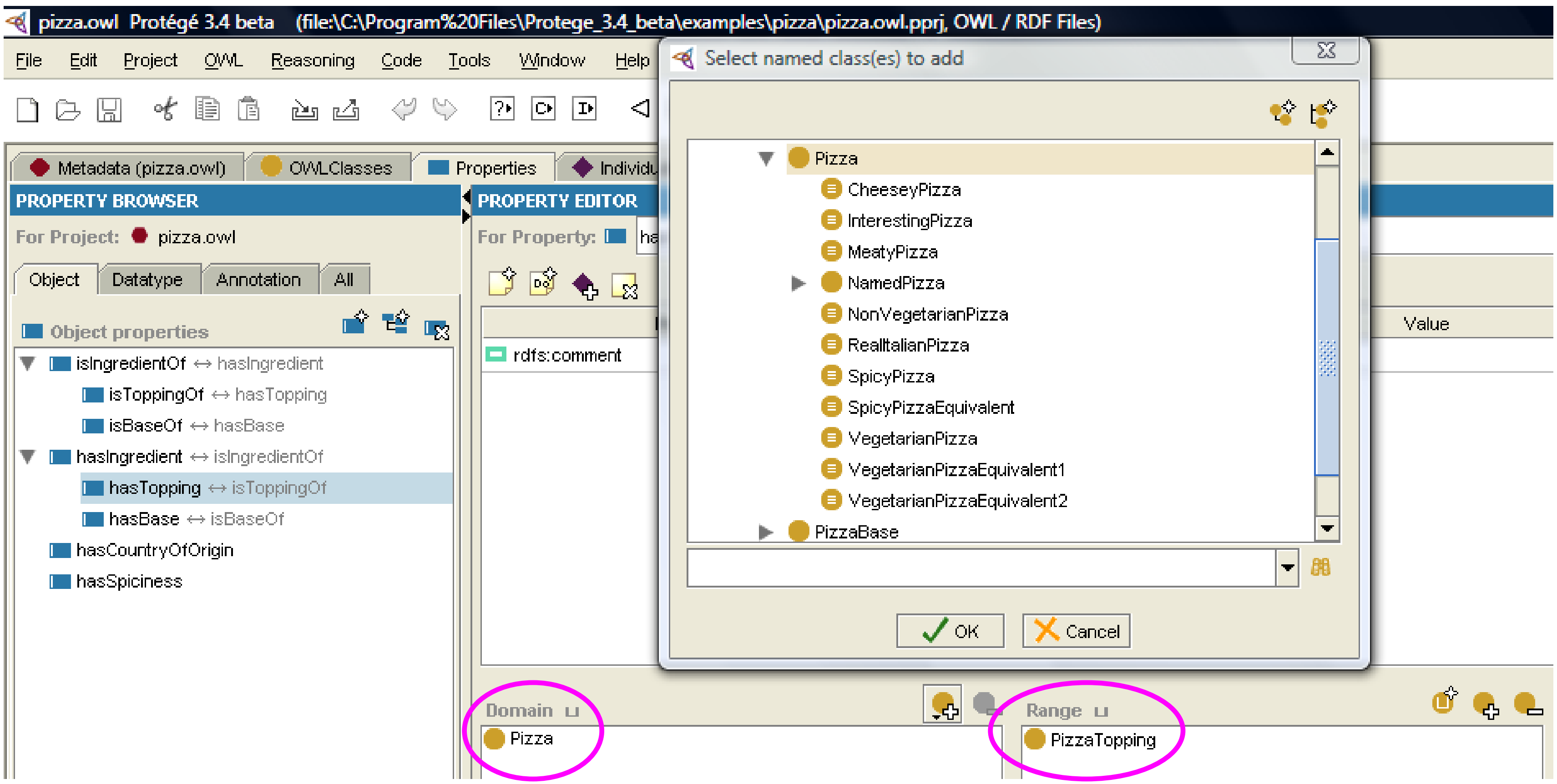
Properties link individuals from the domain to individuals from the range.

Specifiying the **domain (Pizza)** and **range (PizzaTopping)** of hasTopping property.



OWL Properties: Domain and Range (II)

Protege 3.4



Property Restrictions

A **restriction** describes an anonymous class (an unnamed class). The anonymous class 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).

Restrictions are used in OWL class descriptions to specify anonymous superclasses of the class being described.

Existential restrictions describe classes of individuals that participate in at least one relationship along a specified property to individuals that are members of a specified class.

For example, "the class of individuals that have at least one (some) hasTopping relationship to members of MozzarellaTopping".

Universal restrictions describe classes of individuals that for a given property only have relationships along this property to individuals that are members of a specified class.

For example, "the class of individuals that only have hasTopping relationships to members of VegetableTopping".

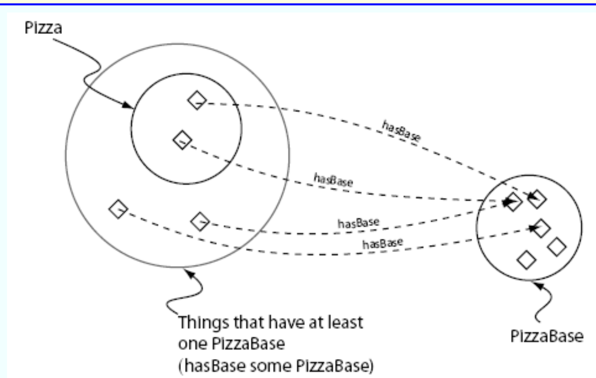
Existential Restrictions (I)

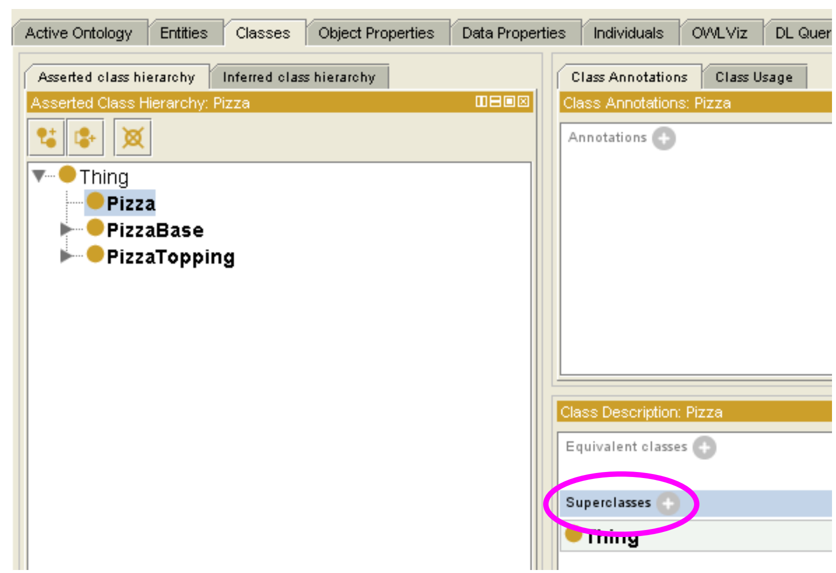
An existential restriction describes a class of individuals that have at least one (some) relationship along a specified property to an individual that is a member of a specified class.

Existential restrictions are also known as Some Restrictions, or as some values from restrictions.

Adding a restriction to Pizza that specifies a Pizza must have a PizzaBase (**hasBase some PizzaBase**). You are creating a *necessary condition*.

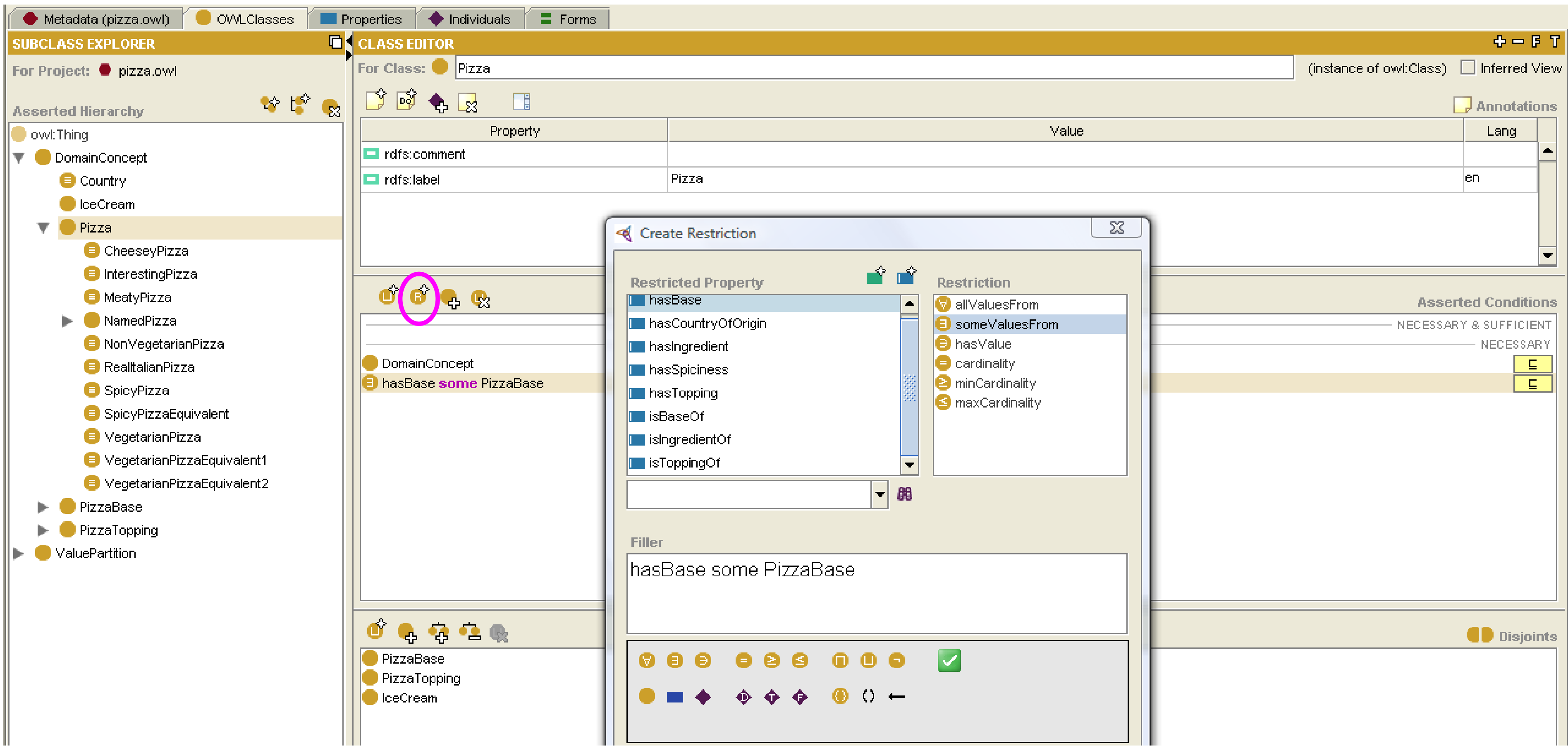
A Pizza is a subclass of the things that have at least one PizzaBase





Existential Restrictions (II)

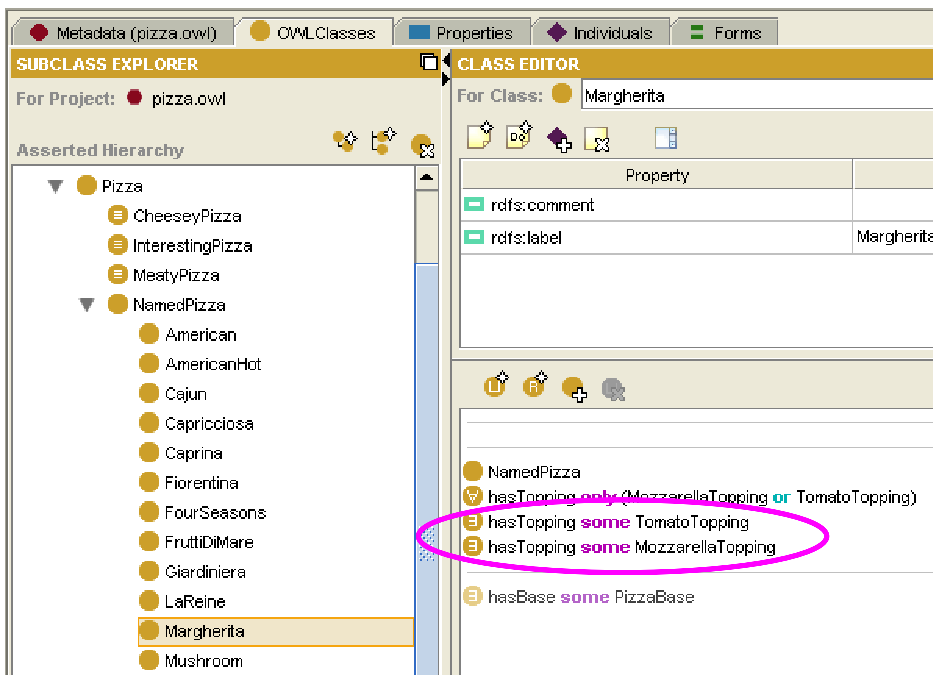
Protege 3.4



Existential Restrictions (III)

Adding two restrictions to say that a MargheritaPizza has the toppings MozzarellaTopping and TomatoTopping.

If something is a member of the class MargheritaPizza it is necessary for it to be a member of: the class NamedPizza, the anonymous class of things that are linked to at least one member of the class MozzarellaTopping via the property hasTopping, and the anonymous class of things that are linked to at least one member of the class TomatoTopping via the property hasTopping.



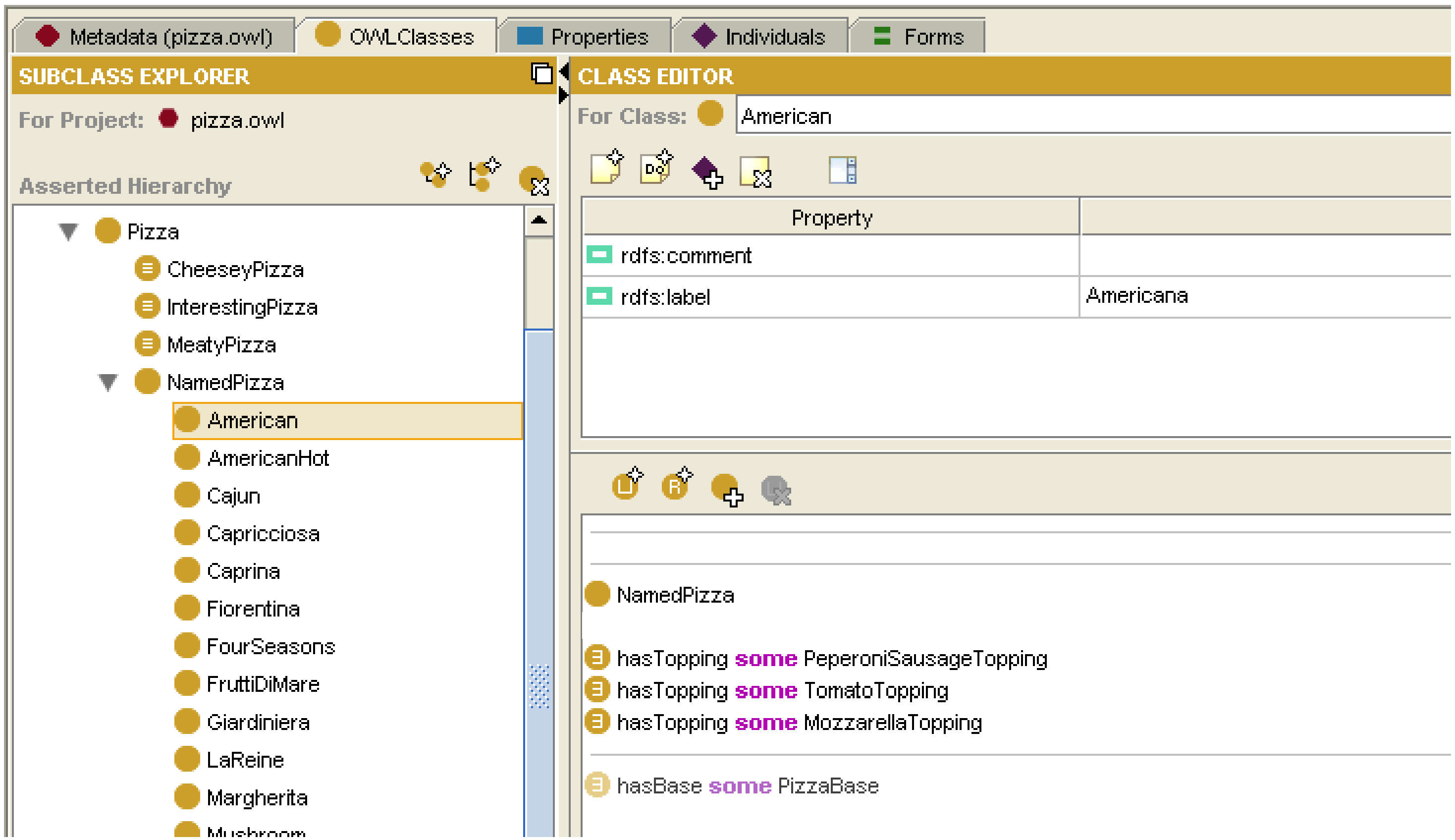
Protege 3.4

Exercise

Create an **American Pizza** that is almost the same as a Margherita Pizza but with an extra topping of pepperoni.

Exercise: Solution

Create an **American Pizza** that is almost the same as a Margherita Pizza but with an extra topping of pepperoni.



Using a Reasoner

One of the key features of ontologies that are described using OWL-DL is that they can be processed by a **reasoner**.

One of the main services offered by a reasoner is to test whether or not one class is a subclass of another class. By performing such tests on the classes in an ontology it is possible for a reasoner to compute the inferred ontology class hierarchy.

Another standard service that is offered by reasoners is 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. A class is deemed to be inconsistent if it cannot possibly have any instances.

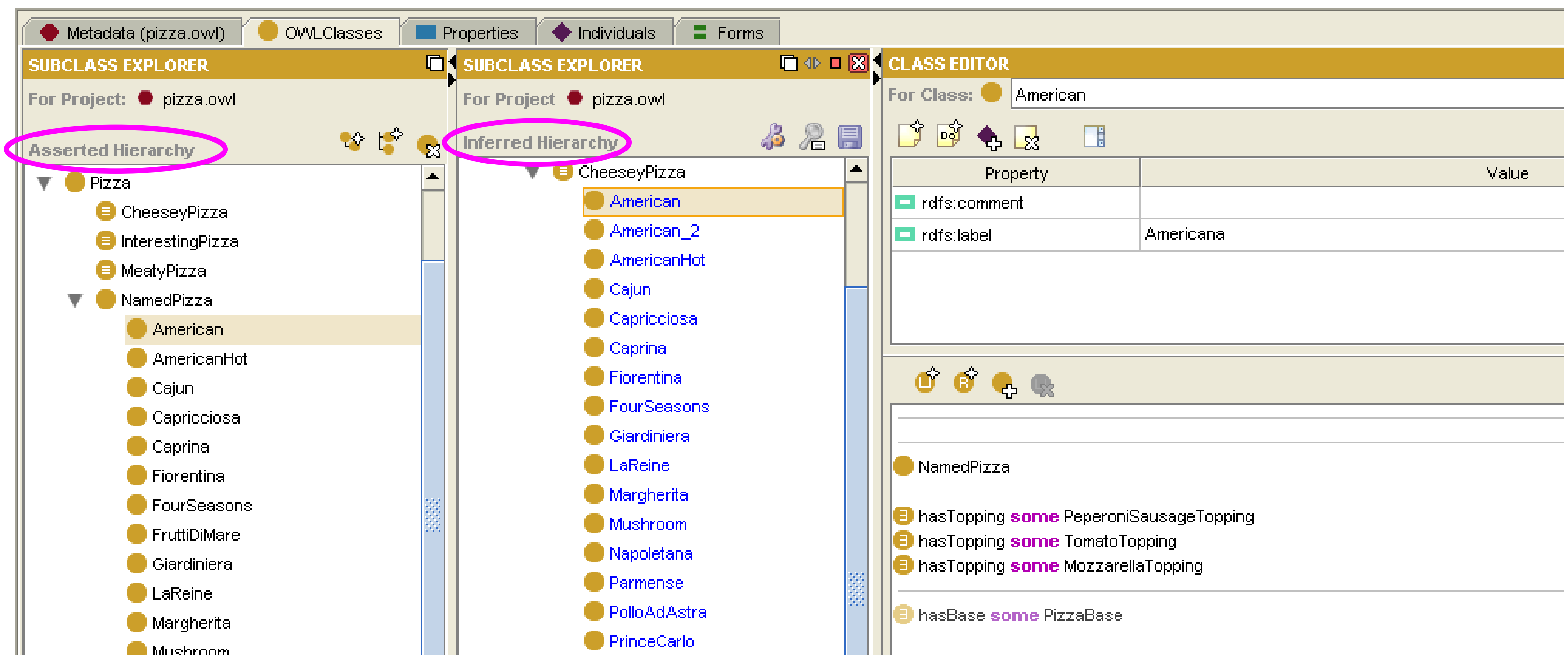
Inferring Ontology Class Hierarchy

The ontology can be ‘sent to the reasoner’ to automatically compute the classification hierarchy.

The ‘manually constructed’ class hierarchy is called the **asserted hierarchy**.

The class hierarchy that is automatically computed by the reasoner is called the **inferred hierarchy**.

Protege 3.4. Pellet 1.5.1

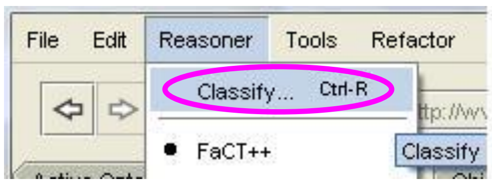


Checking Ontology Consistency

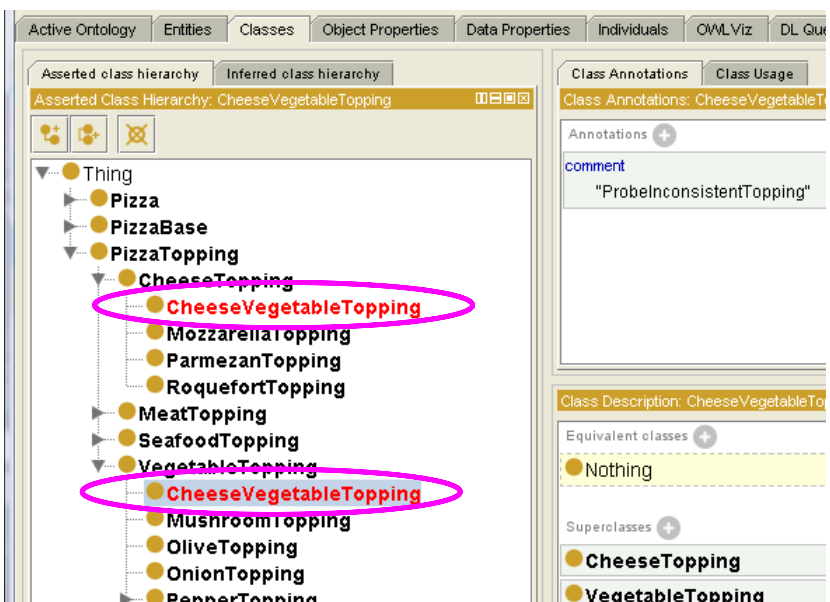
This strategy is often used as a check so that we can see that we have built our ontology correctly.

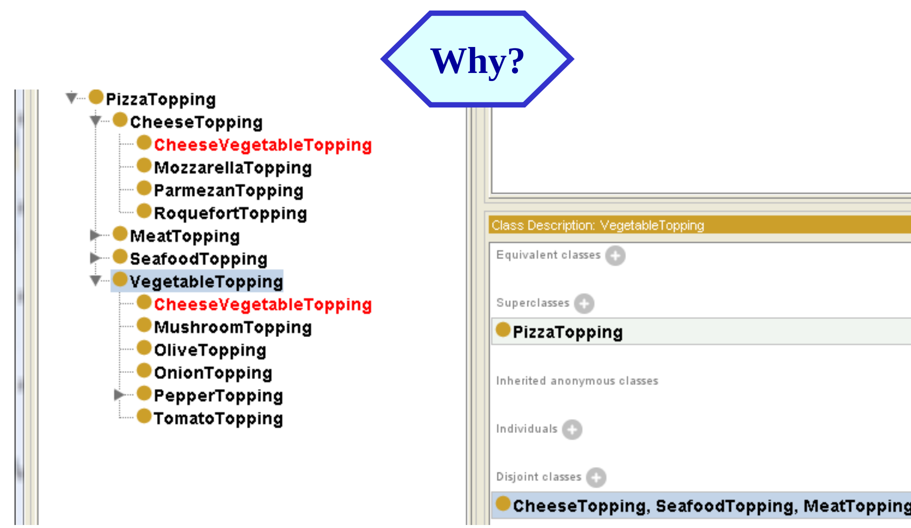
Creating a **CheesyVegetableTopping** as subclass of CheesyTopping and VegetableTopping.

1



2





Necessary and Sufficient Conditions (I)

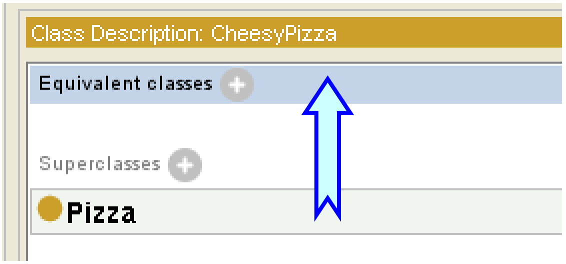
All of the classes that we have created so far have only used necessary conditions to describe them.

Necessary conditions can be read as: "If something is a member of this class then it is necessary to fulfil these conditions".

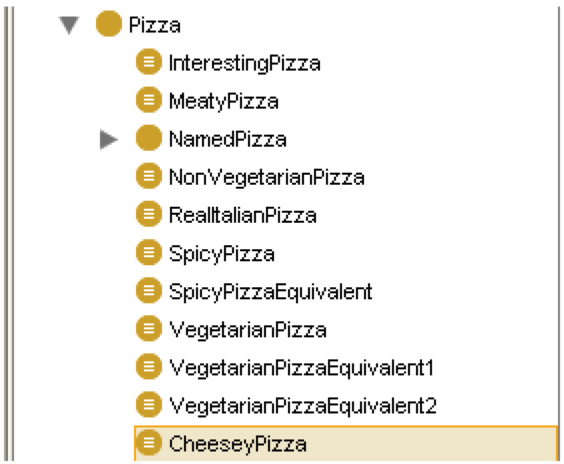
A class that only has necessary conditions is known as a Primitive Class or Partial Class.

With necessary conditions alone, we cannot say that, "*If something fulfils these conditions then it must be a member of this class*". To make this possible we need to change the conditions from necessary conditions to **necessary AND sufficient conditions**.

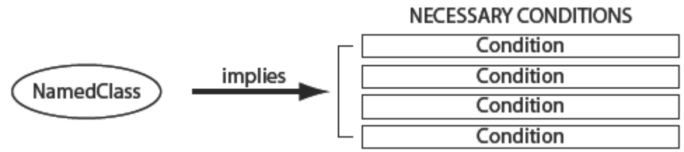
A class that has at least one set of necessary and sufficient conditions is known as a Defined Class or Complete Class.



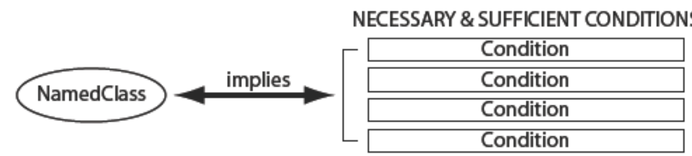
Necessary and Sufficient Conditions (II)



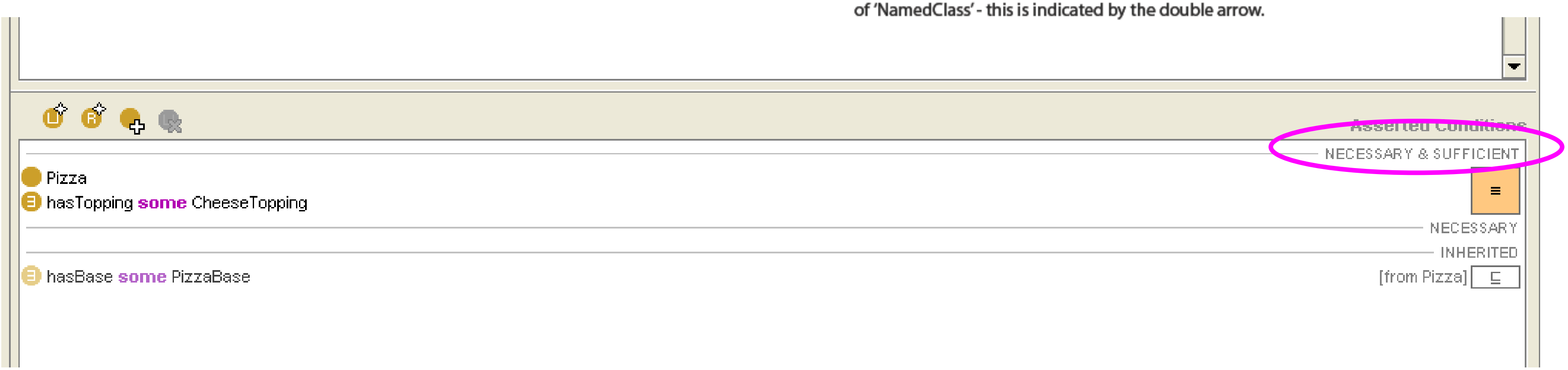
Protege 3.4



If an individual is a member of 'NamedClass' then it must satisfy the conditions. However if some individual satisfies these necessary conditions, we cannot say that it is a member of 'Named Class' (the conditions are not 'sufficient' to be able to say this) - this is indicated by the direction of the arrow.



If an individual is a memeber of 'NamedClass' then it must satisfy the conditions. If some individual satisfies the condtions then the individual must be a member of 'NamedClass' - this is indicated by the double arrow.



Universal Restrictions (I)

All of the restrictions that we have created so far have been existencial ones (some).

However, existential restrictions do not mandate that the only relationships for the given property that can exist must be to individuals that are members of the specified filler class. To restrict the relationships for a given property to individuals that are members of a specific class we must use a **universal restriction**.

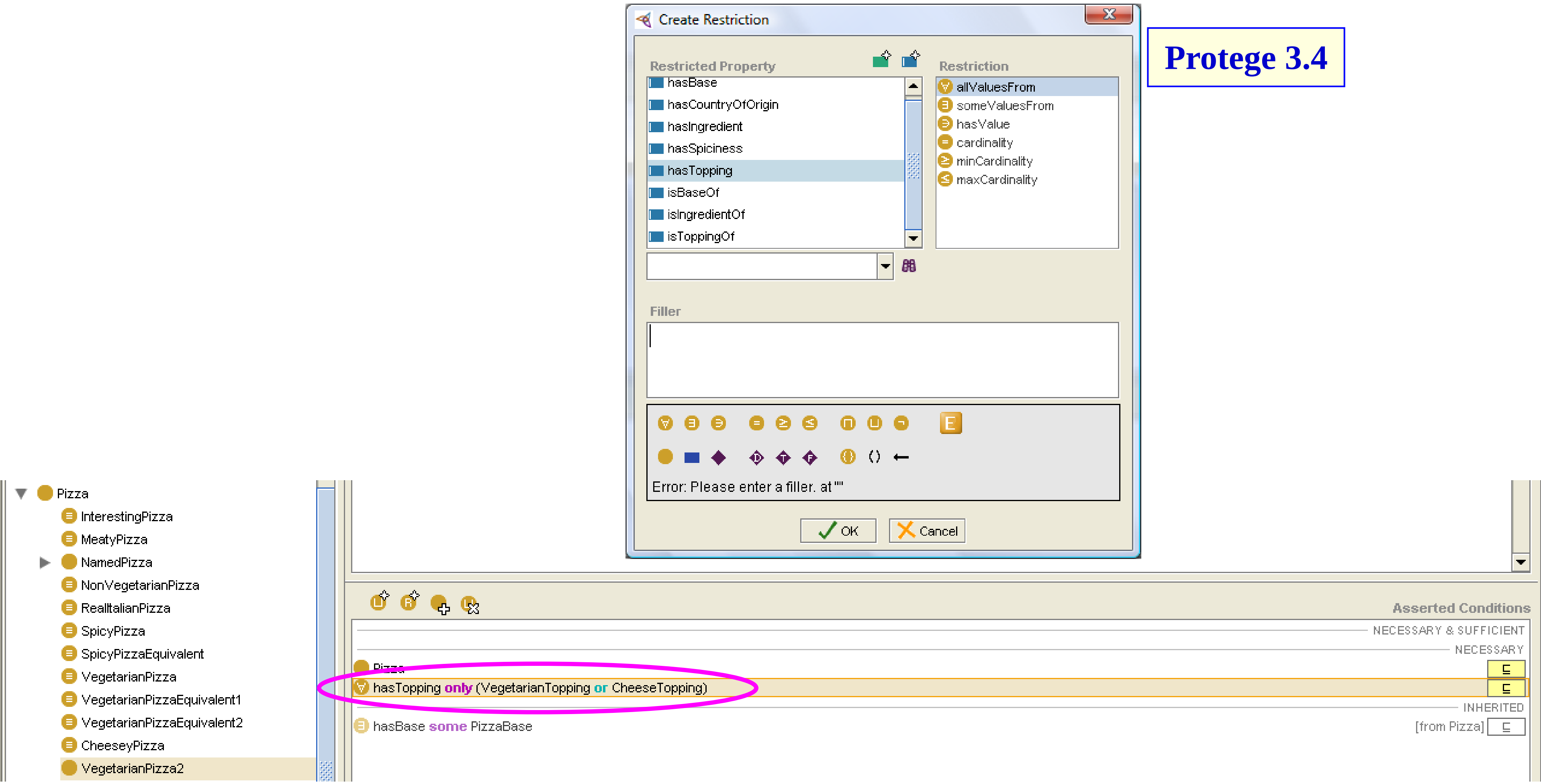
Universal restrictions constrain the relationships along a given property to individuals that are members of a specific class.

For example the universal restriction ∀ hasTopping MozzarellaTopping describes the individuals all of whose hasTopping relationships are to members of the class MozzarellaTopping.

Universal Restrictions (II)

Creating a Vegetarian Pizza that only have toppings that are CheeseTopping or VegetableTopping.

Protege 3.4



Automatic Classification and Open World Assumption (I)

We want to use the reasoner to automatically compute the superclass-subclass relationship (subsumption relationship) between MargheritaPizza and VegetarianPizza.

We believe that MargheritaPizza should be vegetarian pizza (they should be subclasses of VegetarianPizza). This is because they have toppings that are essentially vegetarian toppings — by our definition, vegetarian toppings are members of the classes CheeseTopping or VegetableTopping and their subclasses.

Having previously created a definition for VegetarianPizza (using a set of necessary and sufficient conditions) we can use the reasoner to perform automatic classification and determine the vegetarian pizzas in our ontology.

Automatic Classification and Open World Assumption (II)

MargheritaPizza have not been classified as subclass of VegetarianPizza.

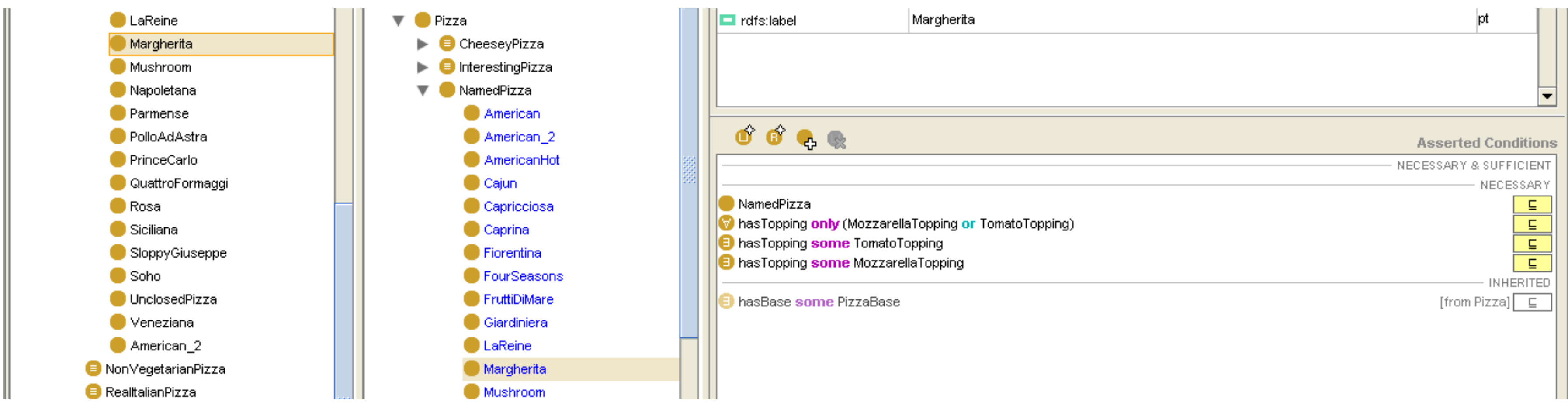
Reasoning in OWL (Description Logics) is based on what is known as the open world assumption (OWA). The open world assumption means that we cannot assume something does not exist until it is explicitly stated that it does not exist.

In the case of our pizza ontology, we have stated that MargheritaPizza has toppings that are kinds of MozzarellaTopping and also kinds of TomatoTopping. Because of the open world assumption, 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.

Automatic Classification and Open World Assumption (III)

To specify explicitly that a MargheritaPizza has toppings that are kinds of MozzarellaTopping or kinds of MargheritaTopping and only kinds of MozzarellaTopping or MargheritaTopping, we must add what is known as a **closure axiom or restriction** on the hasTopping property.

Protege 3.4



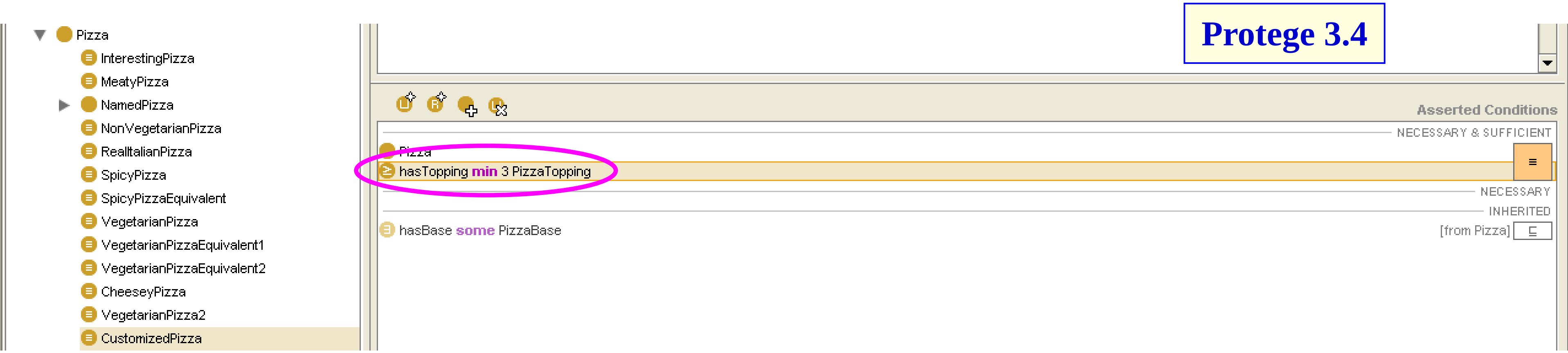
Cardinality Restrictions (I)

In OWL we can describe the class of individuals that have at least, at most or exactly a specified number of relationships with other individuals or datatype values. The restrictions that describe these classes are known as **Cardinality Restrictions**.

- A Minimum Cardinality Restriction specifies the minimum number of P relationships that an individual must participate in.
- A Maximum Cardinality Restriction specifies the maximum number of P relationships that an individual can participate in.
- A Cardinality Restriction specifies the exact number of P relationships that an individual must participate in.

Cardinality Restrictions (II)

Creating a Customized Pizza that has at least three toppings.

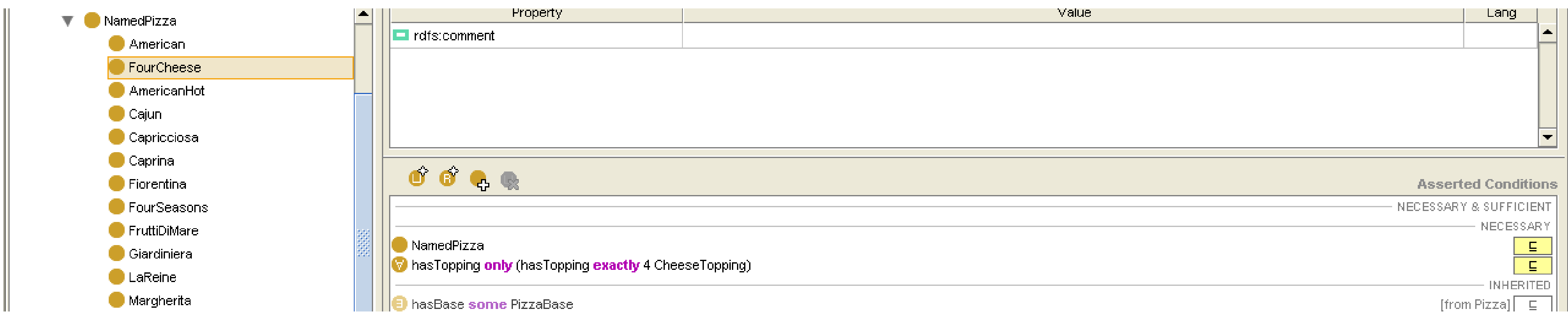


Qualified Cardinality Restrictions (I)

Qualified Cardinality Restrictions (QCR), which are more specific than cardinality restrictions in that they state the class of objects within the restriction.

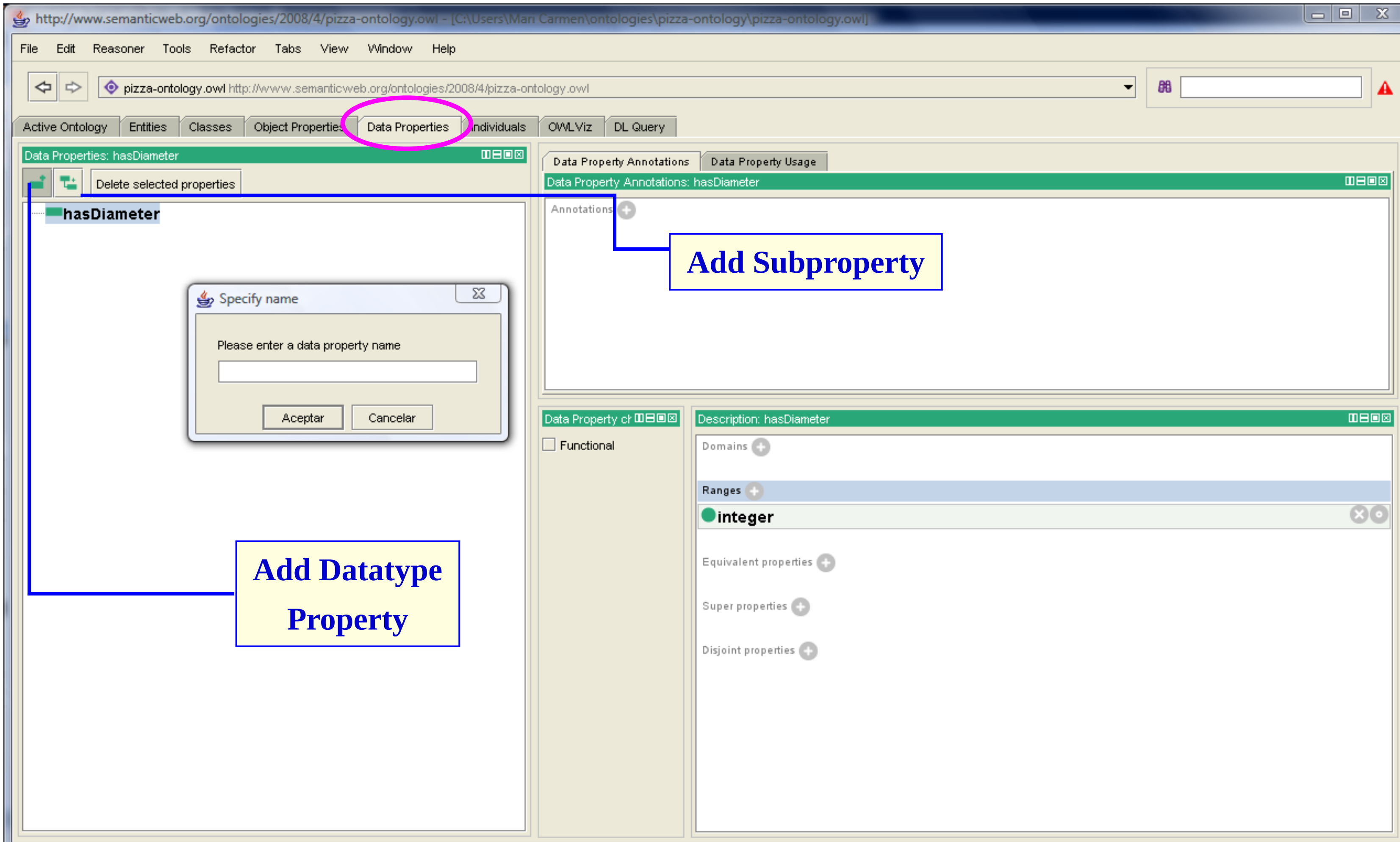
Creating a **Four Cheese Pizza**, as subclass of NamedPizza, which has exactly four cheese toppings.

Protege 3.4



Datatype Properties

Creating a datatype property in the pizza example: hasDiameter.



Restrictions and Boolean Class Constructors

OWL	DL Symbol	Manchester OWL Syntax Keyword	Example
someValuesFrom	$\exists$	some	`hasChild some Man`
allValuesFrom	$\forall$	only	`hasSibling only Woman`
hasValue	$\ni$	value	`hasCountryOfOrigin value England`
minCardinality	$\geq$	min	`hasChild min 3`
cardinality	$=$	exactly	`hasChild exactly 3`
maxCardinality	$\leq$	max	`hasChild max 3`

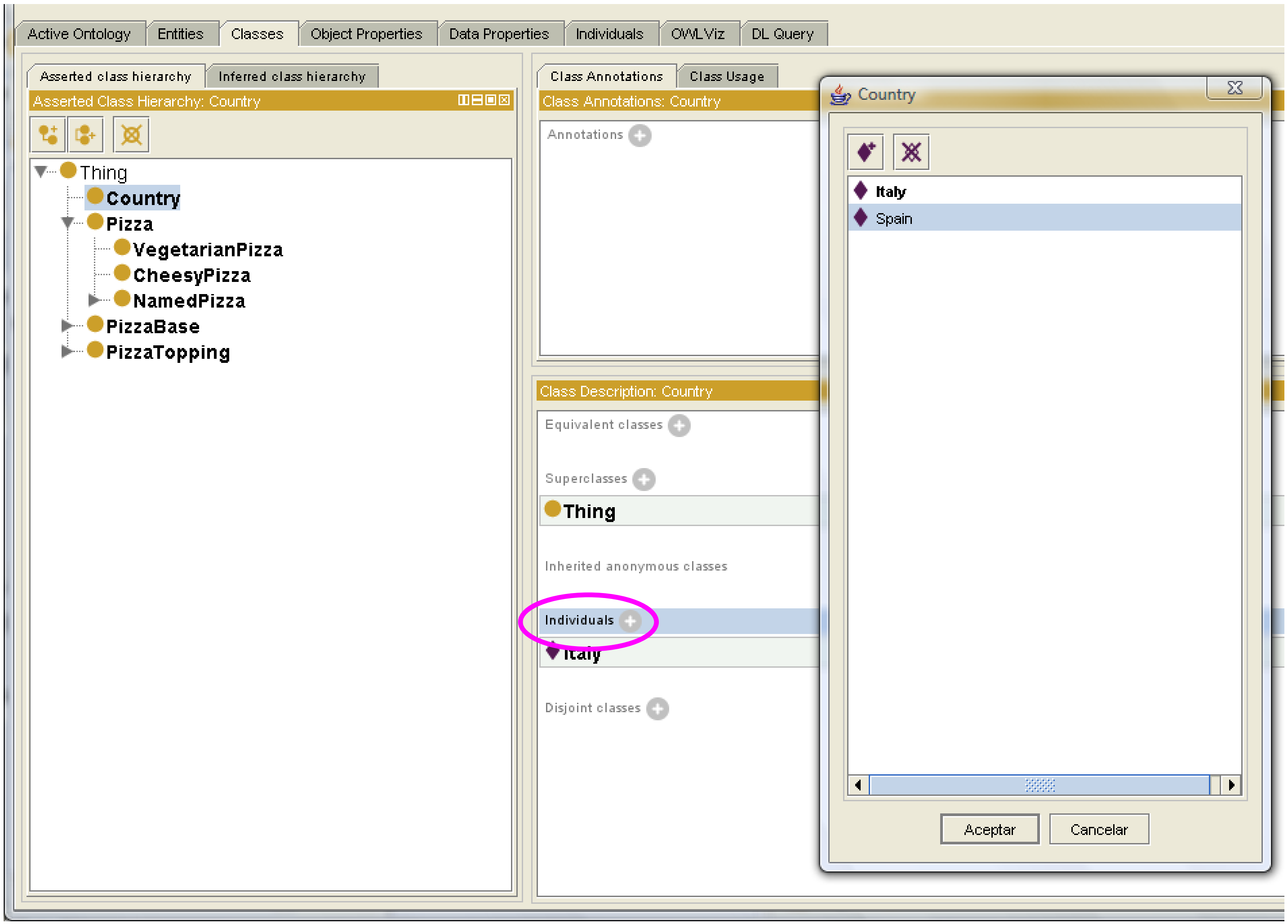
OWL	DL Symbol	Manchester OWL Syntax Keyword	Example
intersectionOf	$\sqcap$	and	`Doctor and Female`
unionOf	$\sqcup$	or	`Man or Woman`
complementOf	$\neg$	not	`not Child`

Exercise

Create a Meaty Pizza.
Create a Vegeterian Pizza, which have no meat and no fish toppings.
Create a Real Italian Pizza, which only have bases that are ThinandCrispy.
Create a subclass of Named Pizza with a topping of Mozzarela.

Individuals

Creating individuals of class Country.



hasValue Restriction

Specifying Italy as country of origin for Mozzarella.