



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.

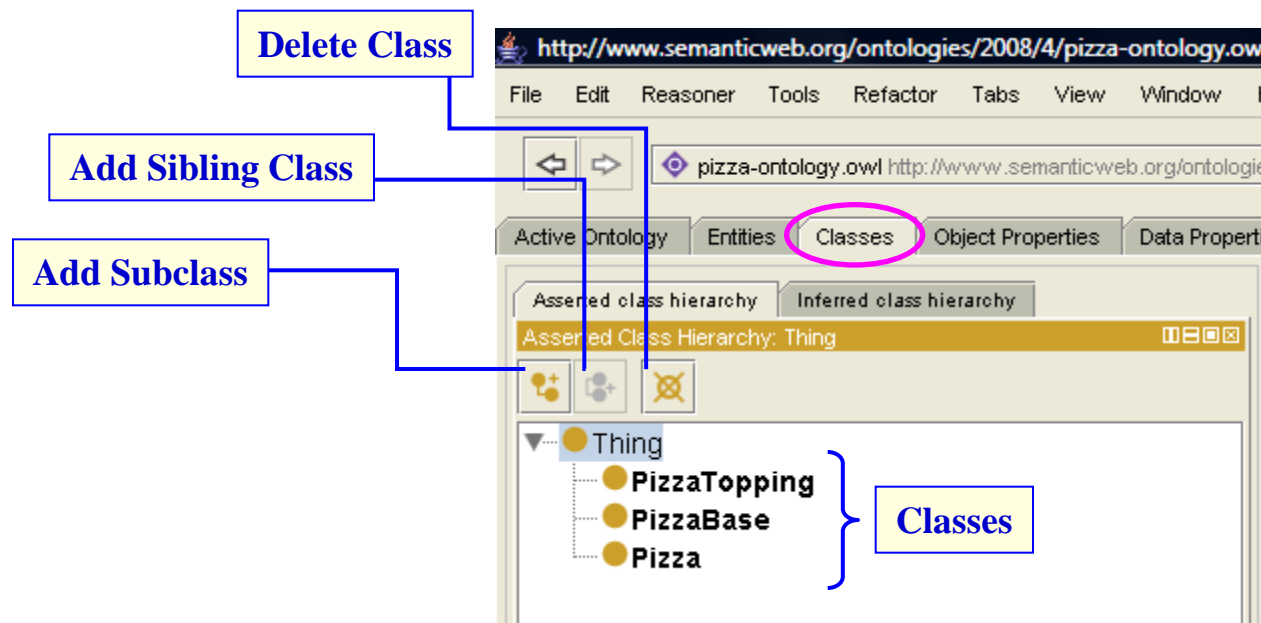
An empty ontology contains one class called *Thing*.

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)

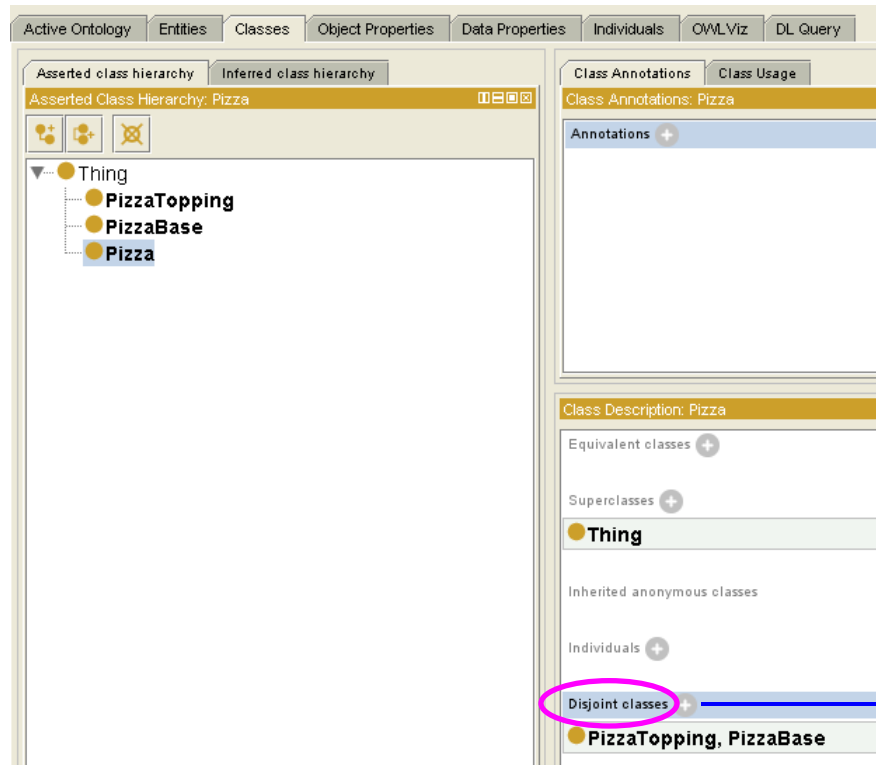
OWL Classes are assumed to ‘**overlap**’. We therefore cannot assume that an individual is not a member of a particular class simply because it has not been asserted to be a member of that class.

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.

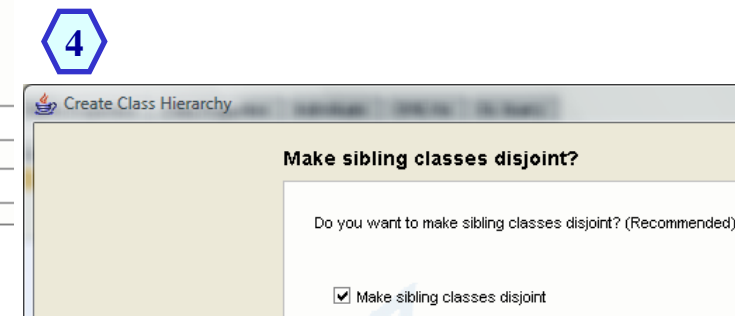
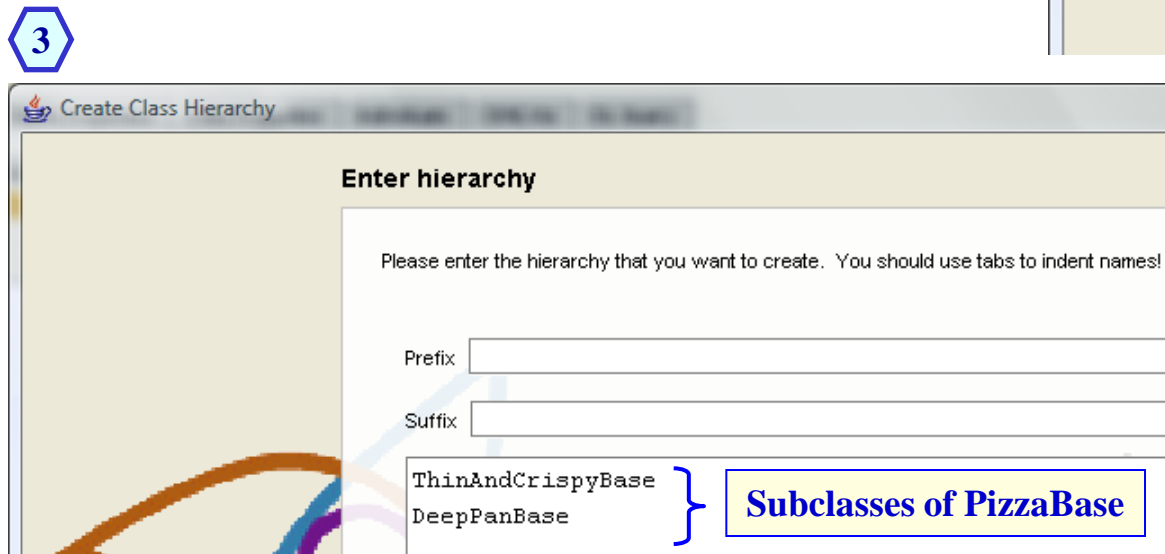
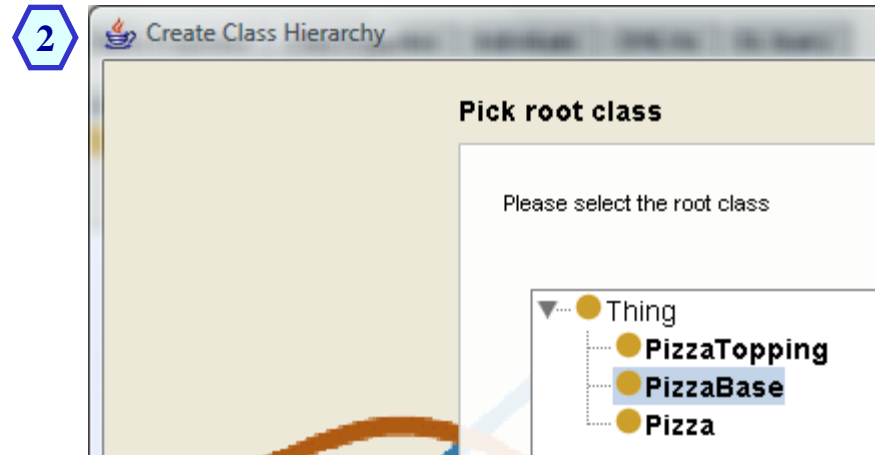
This means that it is not possible for an individual to be a member of a combination of these classes – it would not make sense for an individual to be a Pizza and a PizzaBase.



Add Disjoint Classes

Named Classes (III)

Creating subclasses in the pizza example: **ThinAndCrispyBase** and **DeepPanBase**.



Class Hierarchy (I)

Create Class Hierarchy

Enter hierarchy

Please enter the hierarchy that you want to

Prefix

Suffix

Meat

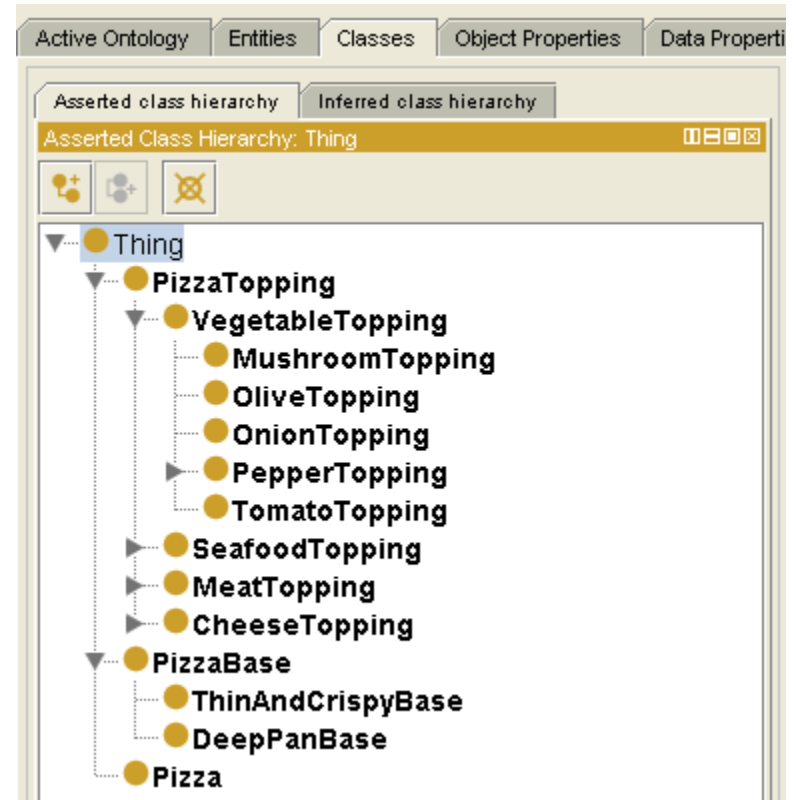
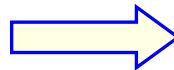
- Roquefort
- Ham
- Pepperoni
- Salami
- SpicyBeef

Seafood

- Anchovy
- Prawn
- Tuna

Vegetable

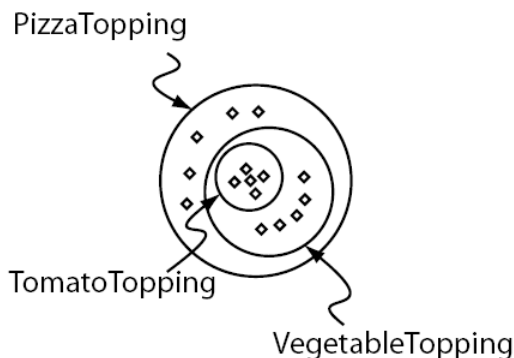
- Mushroom
- Olive
- Onion
- Pepper
- JalapenoPepper
- Tomato



Class Hierarchy (II)

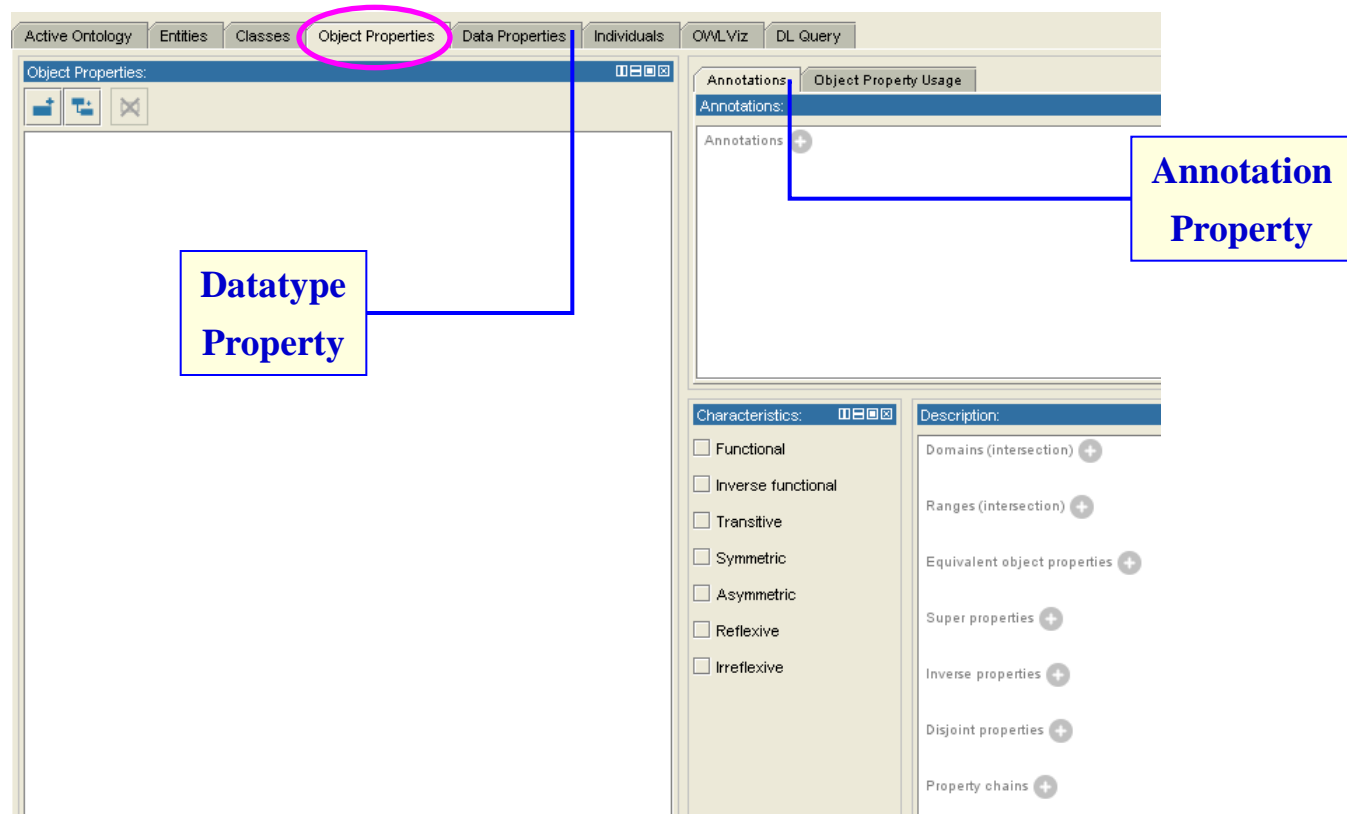
The Meaning of Subclass: all individuals that are members of the class TomatoTopping are members of the class VegetableTopping and members of the class PizzaTopping as we have stated that TomatoTopping is a subclass of VegetableTopping which is a subclass of PizzaTopping.

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**.

The screenshot displays the Protégé ontology editor interface. The 'Object Properties' tab is active, showing a list of properties: **hasIngredient**, **hasBase**, and **hasTopping**. A bracket groups these three properties under the label 'Object Properties'. The 'hasBase' property is selected, and its annotations are visible on the right. The 'Characteristics' panel on the bottom right shows various property constraints like Functional, Transitive, etc. The 'Description' panel on the bottom right shows the domain and range of the property.

Delete Object Property

Add Subproperty

Add Object Property

Object Properties

Annotations: hasBase

Characteristics: hasBase

Description: hasBase

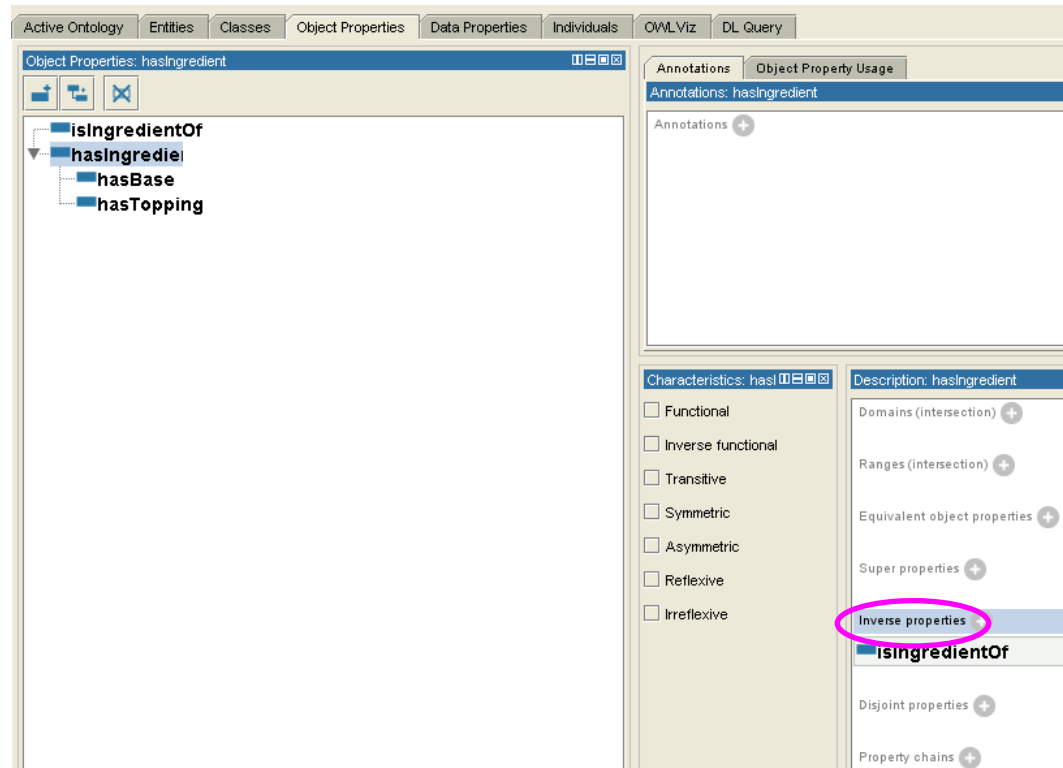
- Domains (intersection) +
- Ranges (intersection) +
- Equivalent object properties +
- Super properties +
- hasIngredient**
- Inverse properties +
- Disjoint properties +
- Property chains +

Inverse Properties

Each object property may have a corresponding **inverse property**.

If some property links individual a to individual b, then its inverse property links 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

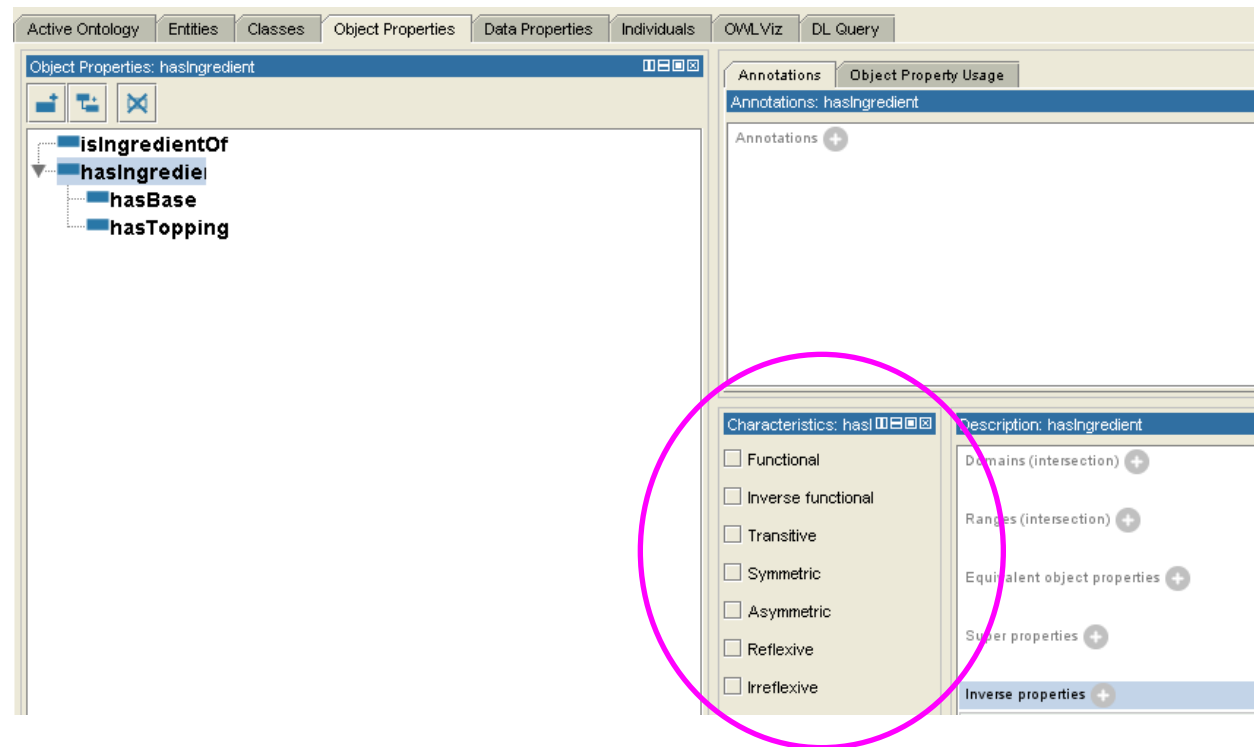
Transitive Properties

Symmetric Properties

Antisymmetric Properties

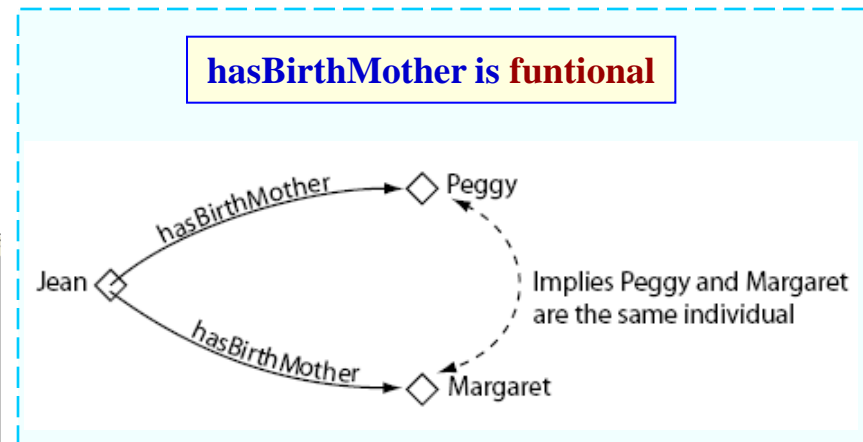
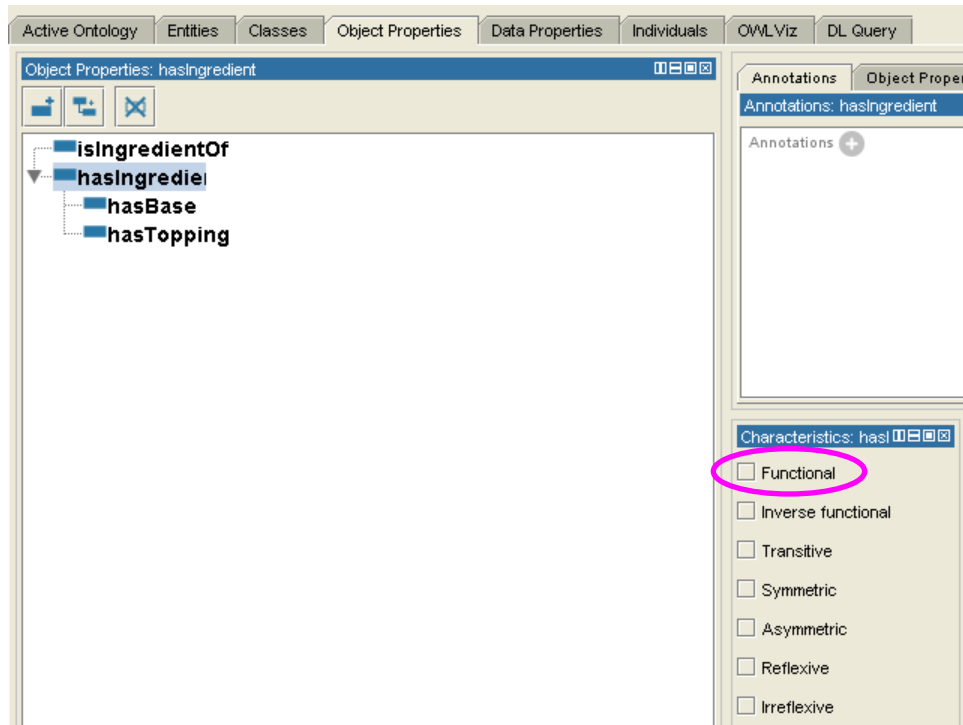
Reflexive Properties

Irreflexive Properties



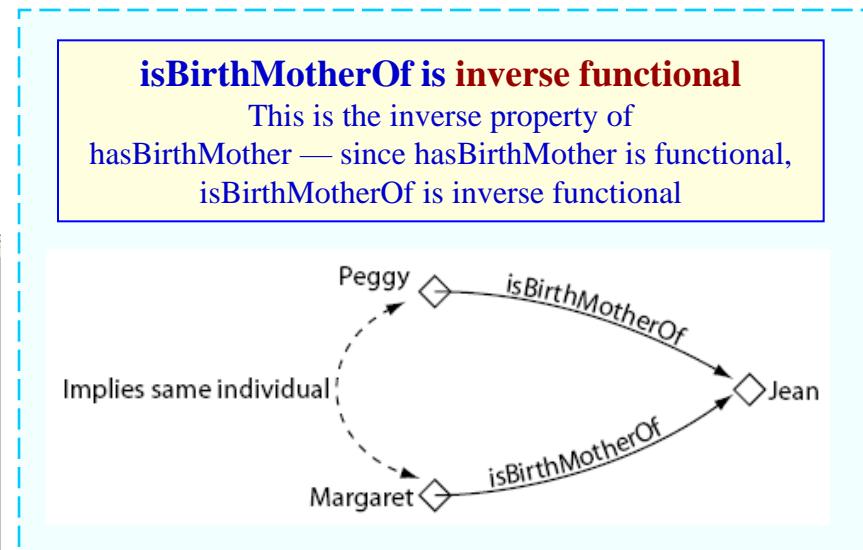
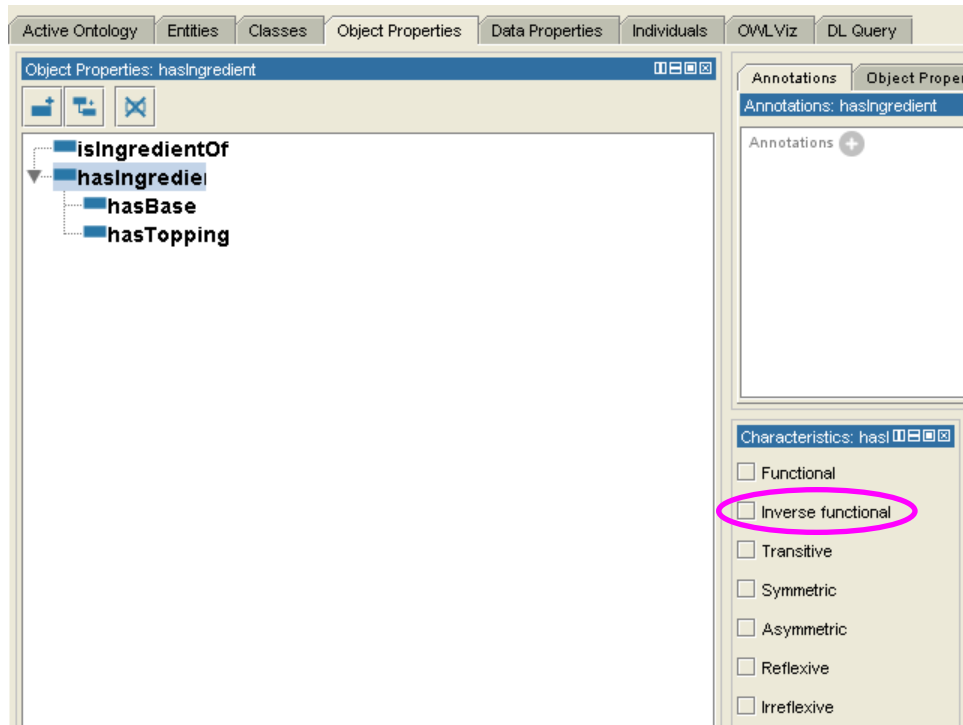
Functional Properties

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.



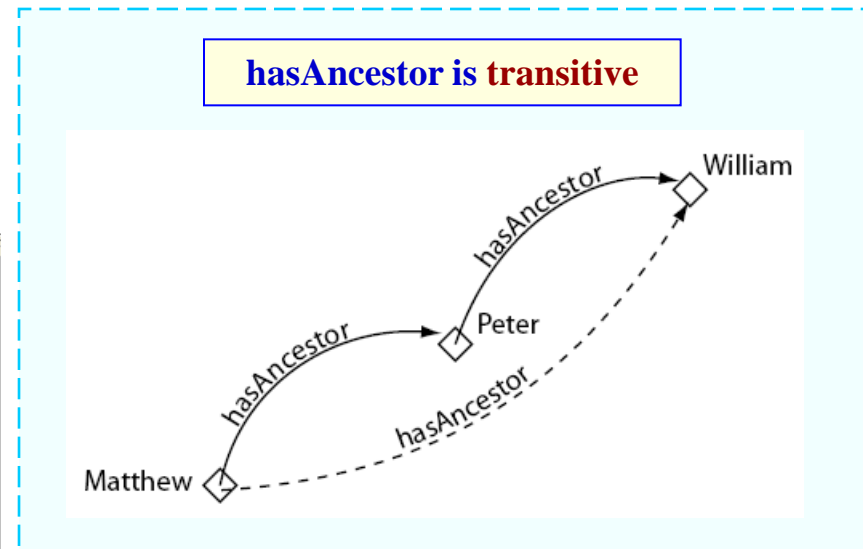
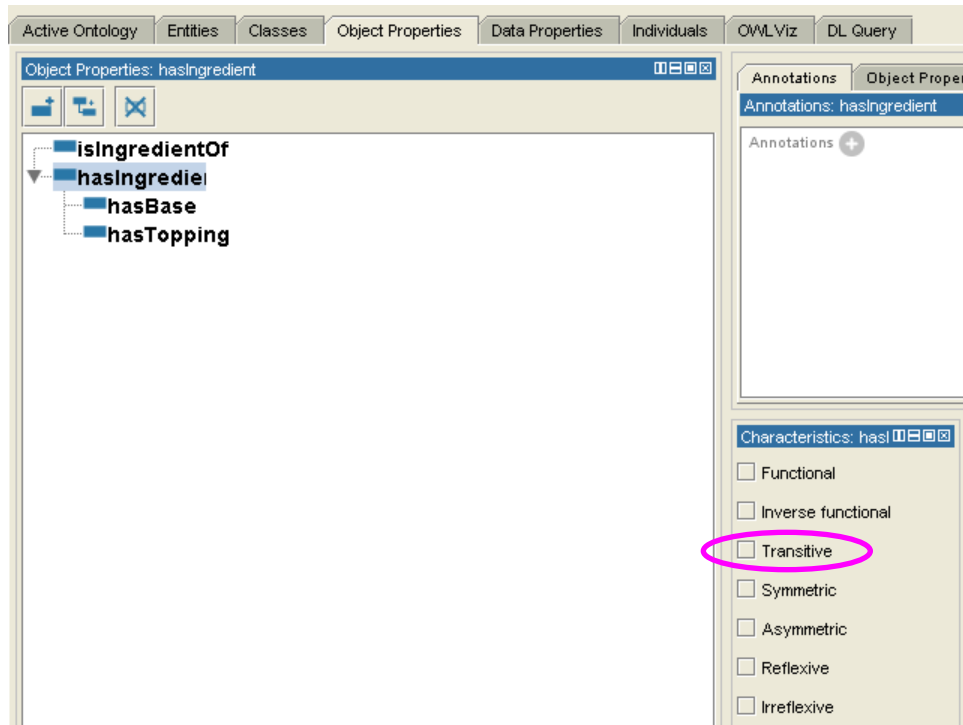
Inverse Functional Properties

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.



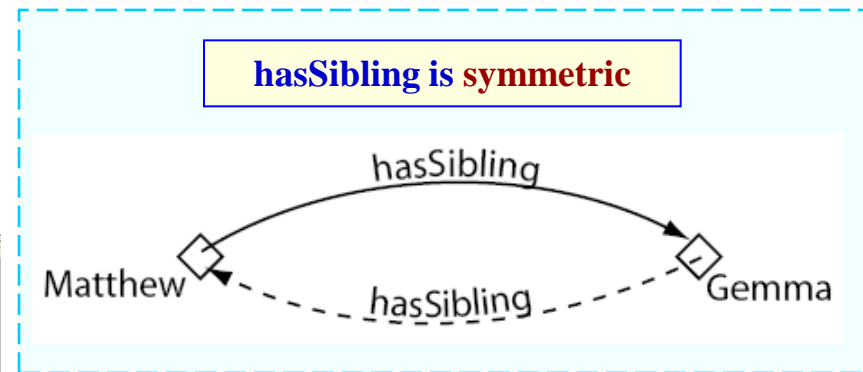
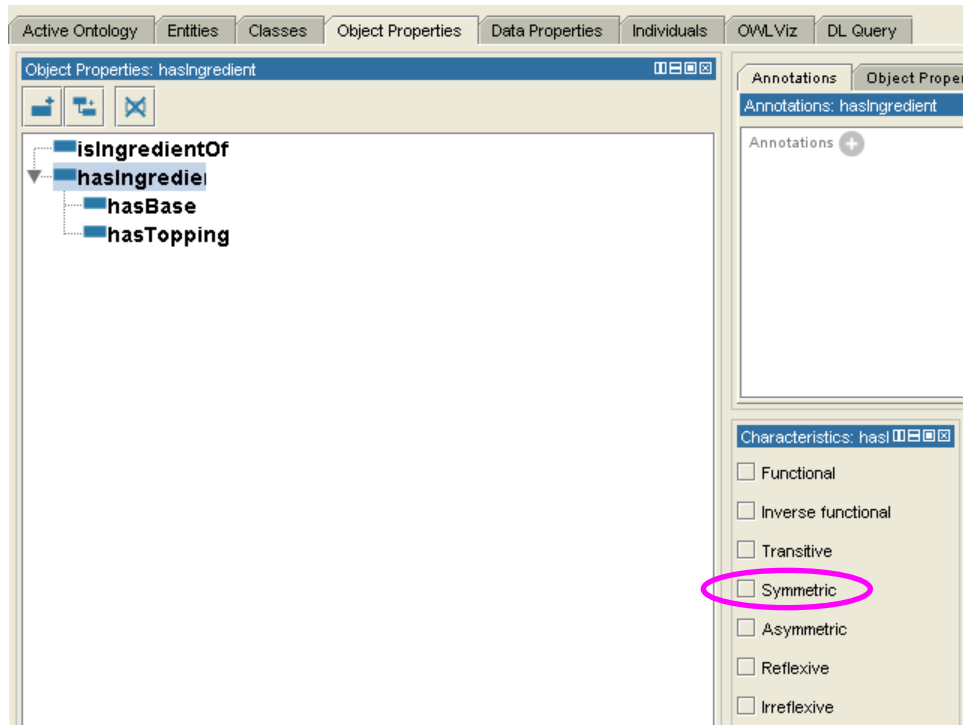
Transitive Properties

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.



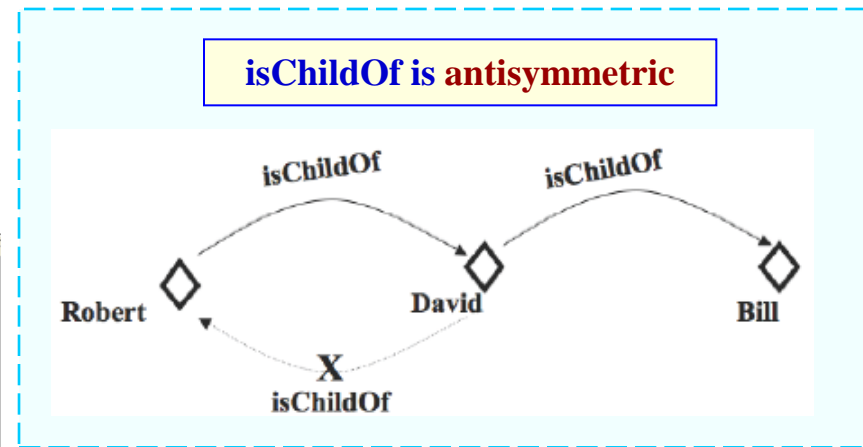
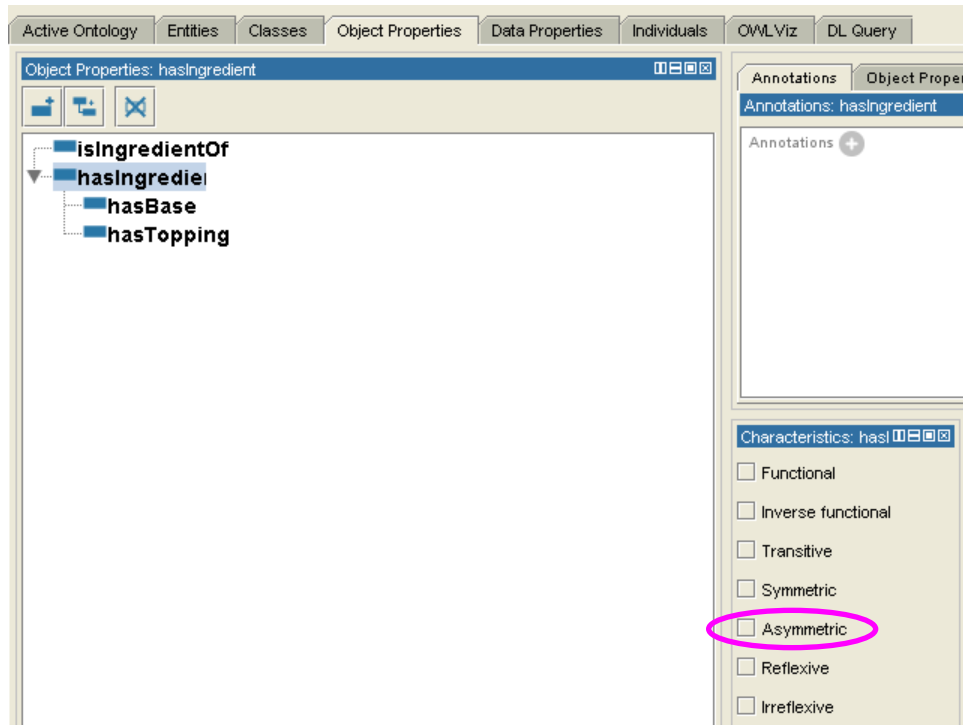
Symmetric Properties

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.



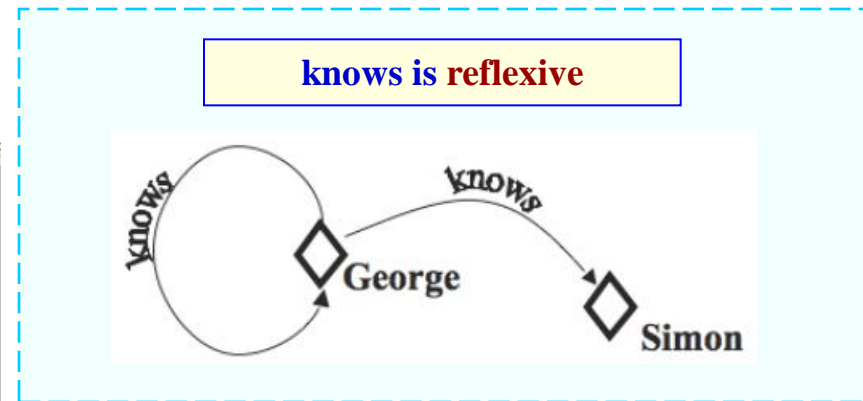
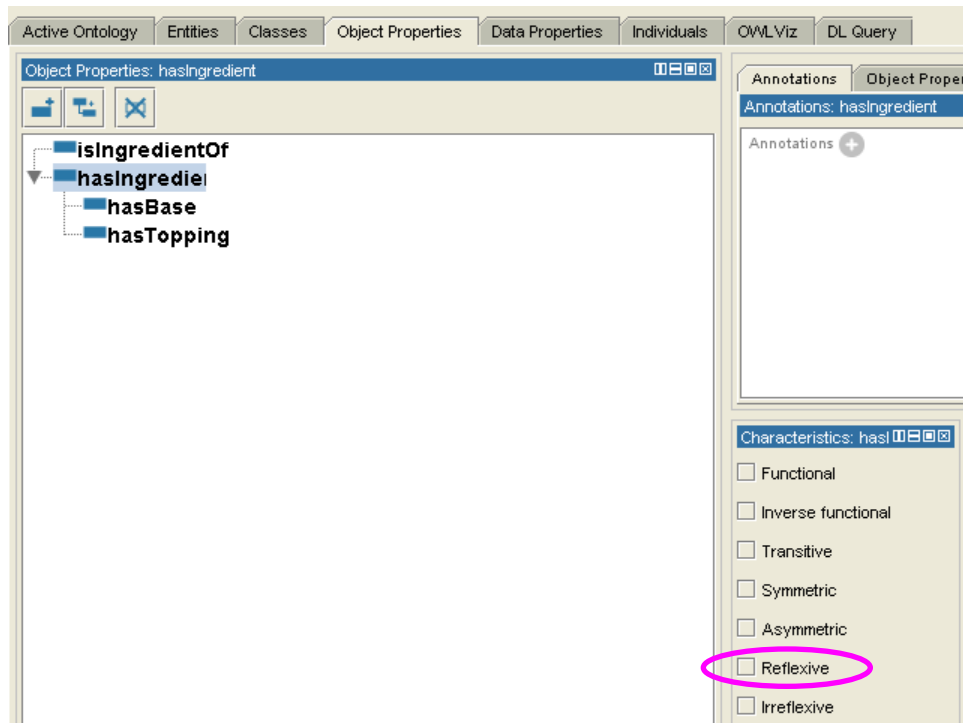
Antisymmetric Properties

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.



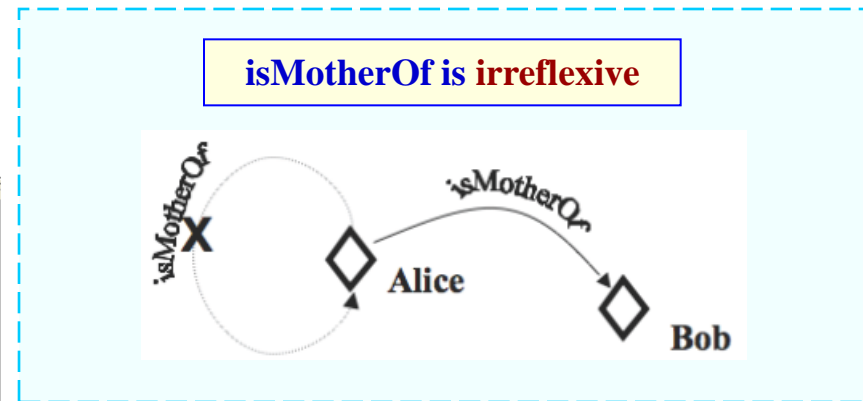
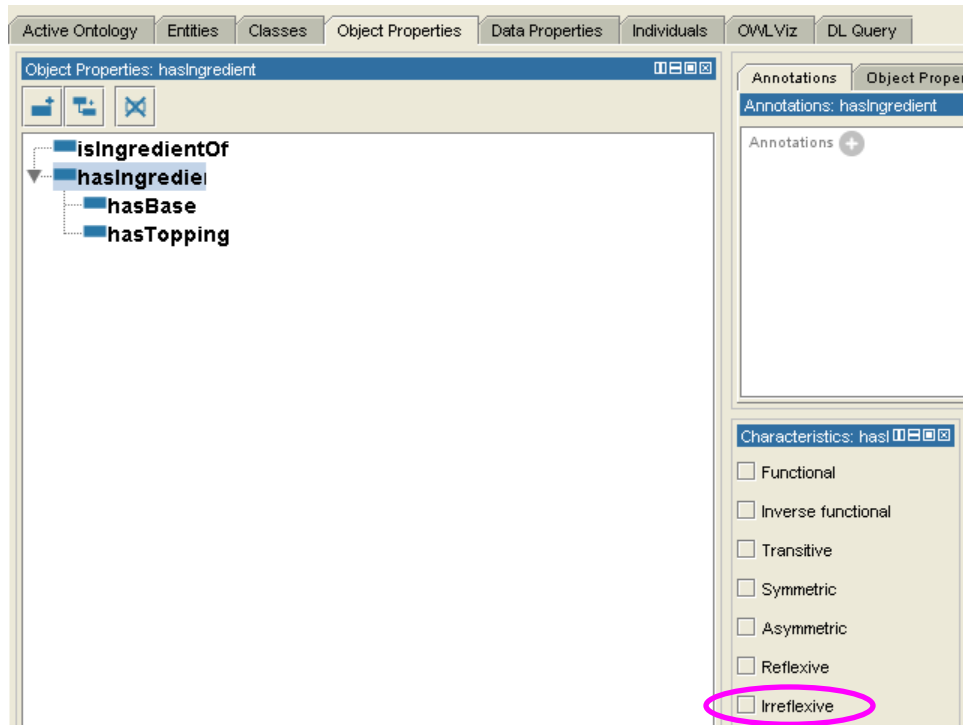
Reflexive Properties

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



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



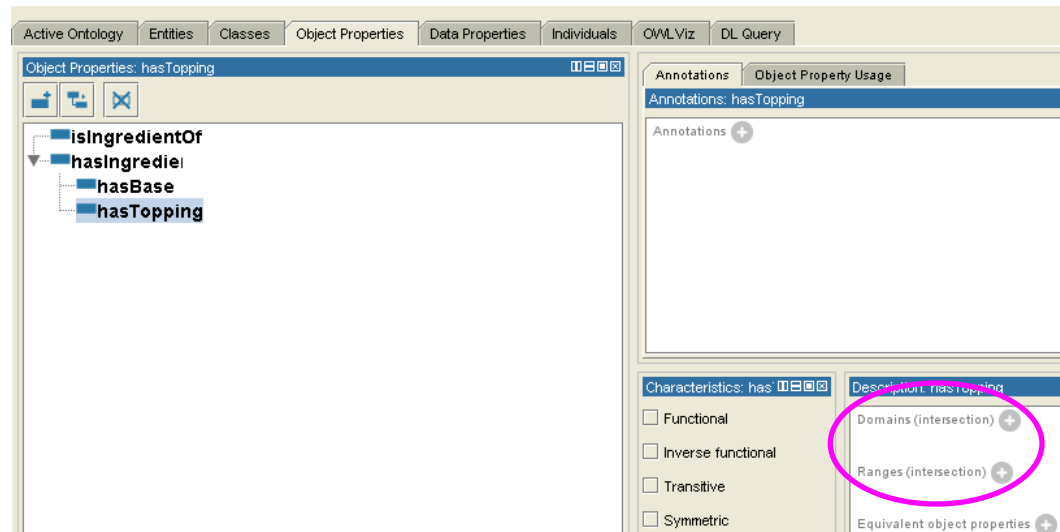
Modelling that a pizza has only one pizza base; and that if a pizza topping has ingredients, then the pizza itself contains also such ingredients.

OWL Properties: Domain and Range (I)

Properties may have a **domain** and a **range** specified.

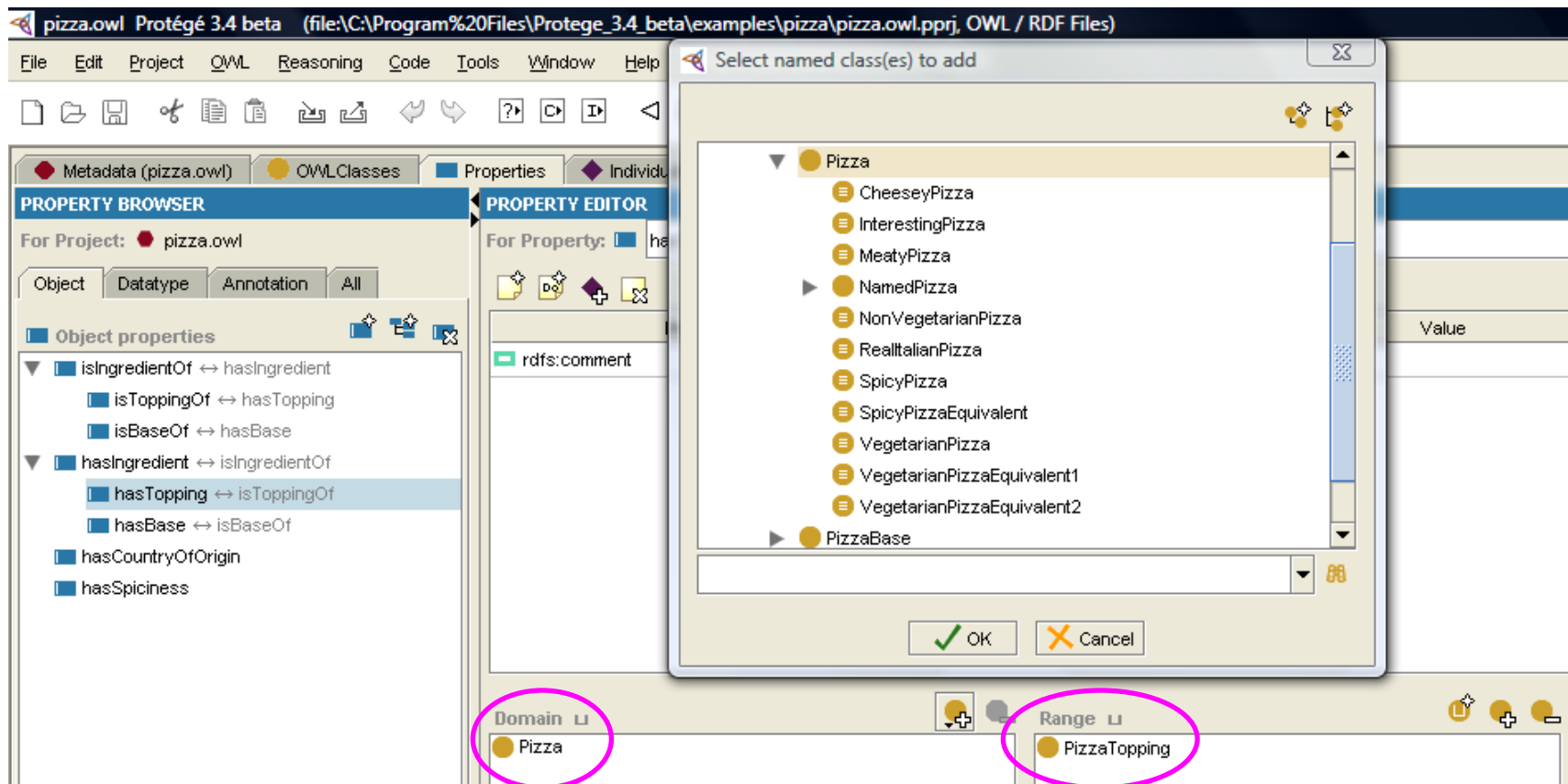
Properties link individuals from the domain to individuals from the range.

Specifying 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.

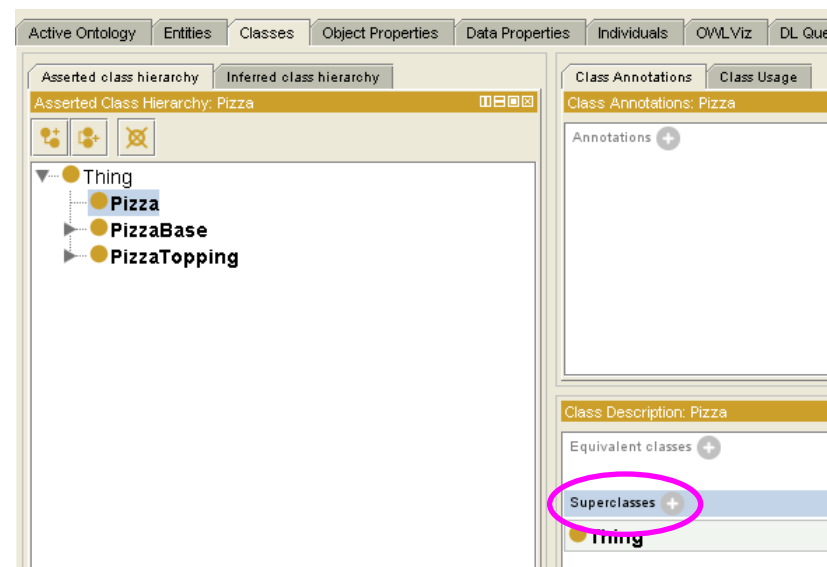
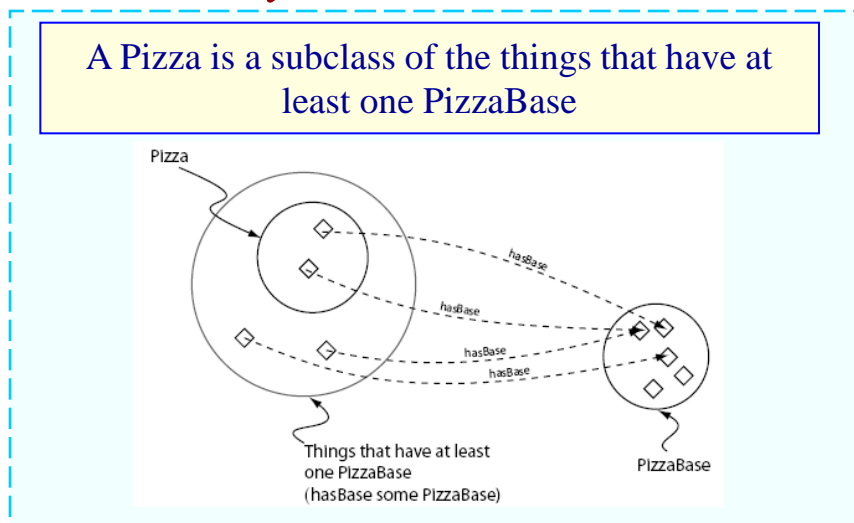
For example, “the class of individuals that **only** have hasTopping relationships to members of VegetableTopping”.

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*.



Existential Restrictions (II)

Protege 3.4

The screenshot shows the Protege 3.4 interface with the 'CLASS EDITOR' for the 'Pizza' class. The 'SUBCLASS EXPLORER' on the left shows the hierarchy of classes, including 'Pizza' and its subclasses. The 'CLASS EDITOR' displays the 'Pizza' class with its properties and values. A 'Create Restriction' dialog box is open, showing the 'Restricted Property' list with 'hasBase' selected. The 'Restriction' list includes 'someValuesFrom'. The 'Filler' field contains 'hasBase some PizzaBase'. The 'Asserted Conditions' section shows 'NECESSARY & SUFFICIENT' and 'NECESSARY' options. The 'Disjoints' section is also visible.

Metadata (pizza.owl) | OWLClasses | Properties | Individuals | Forms

SUBCLASS EXPLORER
For Project: pizza.owl

Asserted Hierarchy

- owl:Thing
 - DomainConcept
 - Country
 - IceCream
 - Pizza
 - CheeseyPizza
 - InterestingPizza
 - MeatyPizza
 - NamedPizza
 - NonVegetarianPizza
 - RealItalianPizza
 - SpicyPizza
 - SpicyPizzaEquivalent
 - VegetarianPizza
 - VegetarianPizzaEquivalent1
 - VegetarianPizzaEquivalent2
 - PizzaBase
 - PizzaTopping
 - ValuePartition

CLASS EDITOR
For Class: Pizza (instance of owl:Class) ☐ Inferred View

Property	Value	Lang
rdfs:comment		
rdfs:label	Pizza	en

Create Restriction

Restricted Property

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

Restriction

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

Filler

hasBase some PizzaBase

Asserted Conditions

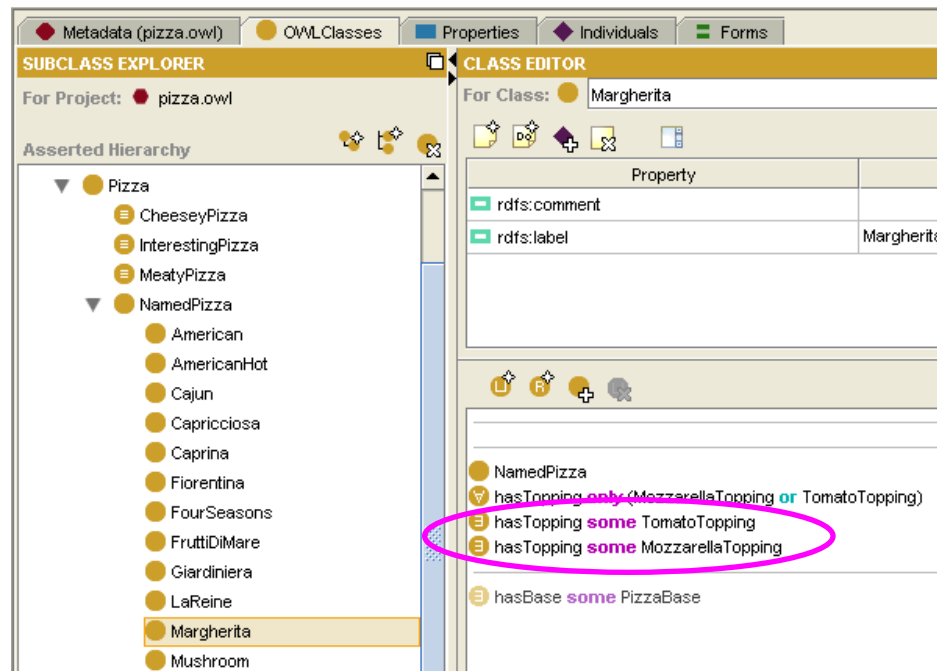
- NECESSARY & SUFFICIENT
- NECESSARY

Disjoints

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.

The screenshot shows the Protégé ontology editor interface. The top tabs are Metadata (pizza.owl), OWLClasses, Properties, Individuals, and Forms. The left pane, titled 'SUBCLASS EXPLORER', shows the 'Asserted Hierarchy' for the project 'pizza.owl'. It displays a tree structure where 'Pizza' is the root, with subclasses 'CheeseyPizza', 'InterestingPizza', 'MeatyPizza', and 'NamedPizza'. 'NamedPizza' is expanded, showing 'American' as a subclass, which is currently selected. Other subclasses of 'NamedPizza' include 'AmericanHot', 'Cajun', 'Capricciosa', 'Caprina', 'Fiorentina', 'FourSeasons', 'FruttiDiMare', 'Giardiniera', 'LaReine', 'Margherita', and 'Mushroom'. The right pane, titled 'CLASS EDITOR', shows the configuration for the 'American' class. It includes a table for properties and a list of restrictions.

Property	
rdfs:comment	
rdfs:label	Americana

Below the table, there are icons for adding and removing restrictions. The restrictions listed are:

- NamedPizza
- hasTopping **some** PeperoniSausageTopping
- hasTopping **some** TomatoTopping
- hasTopping **some** MozzarellaTopping
- hasBase **some** PizzaBase

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

The screenshot displays the Protege 3.4.1 interface with the Pellet 1.5.1 reasoner. The 'SUBCLASS EXPLORER' window is open for the project 'pizza.owl'. It shows two panes: 'Asserted Hierarchy' and 'Inferred Hierarchy'. The 'Asserted Hierarchy' pane shows a tree structure starting with 'Pizza', which includes subclasses like 'CheeseyPizza', 'InterestingPizza', 'MeatyPizza', and 'NamedPizza'. The 'NamedPizza' class is expanded, showing subclasses: 'American', 'AmericanHot', 'Cajun', 'Capricciosa', 'Caprina', 'Fiorentina', 'FourSeasons', 'FruttiDiMare', 'Gardiniera', 'LaReine', 'Margherita', and 'Mushroom'. The 'Inferred Hierarchy' pane shows a similar tree structure starting with 'CheeseyPizza', which includes subclasses like 'American', 'American_2', 'AmericanHot', 'Cajun', 'Capricciosa', 'Caprina', 'Fiorentina', 'FourSeasons', 'Gardiniera', 'LaReine', 'Margherita', 'Mushroom', 'Napoletana', 'Parmense', 'PolloAdAstra', and 'PrinceCarlo'. The 'CLASS EDITOR' window is open for the class 'American', showing a table with properties and values. The table has two columns: 'Property' and 'Value'. The first row shows 'rdfs:comment' with an empty value. The second row shows 'rdfs:label' with the value 'Americana'. Below the table, there are icons for 'L' (Left), 'R' (Right), and a plus sign. The 'NamedPizza' class is also visible in the class editor, showing its subclasses: 'hasTopping some PeperoniSausageTopping', 'hasTopping some TomatoTopping', 'hasTopping some MozzarellaTopping', and 'hasBase some PizzaBase'.

Property	Value
rdfs:comment	
rdfs:label	Americana

NamedPizza

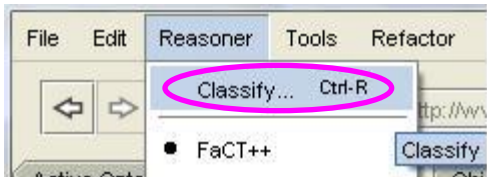
- hasTopping some PeperoniSausageTopping
- hasTopping some TomatoTopping
- hasTopping some MozzarellaTopping
- hasBase some PizzaBase

Checking Ontology Consistency

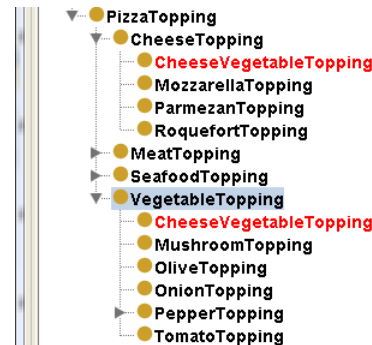
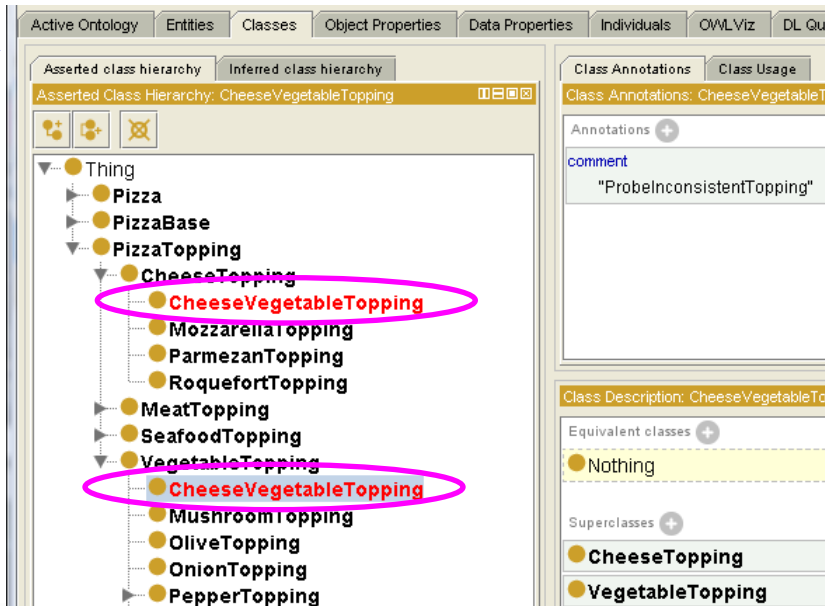
This strategy is often used as a **check** so that we can see that we have built our ontology correctly.

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

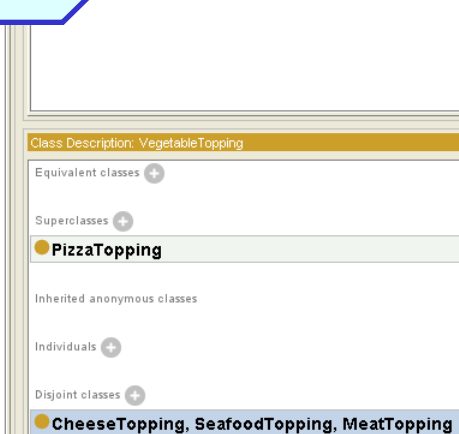
1



2



Why?



Necessary and Sufficient Conditions (I)

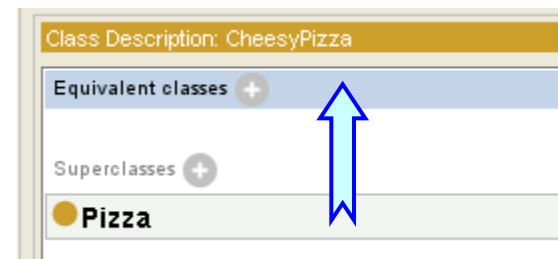
All of the classes that we have created so far have only used necessary conditions to describe them.

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**.

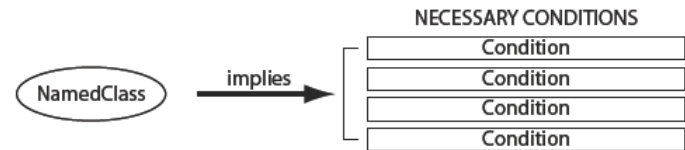
A class that has at least one set of necessary and sufficient conditions is known as a **Defined Class or Complete Class**.



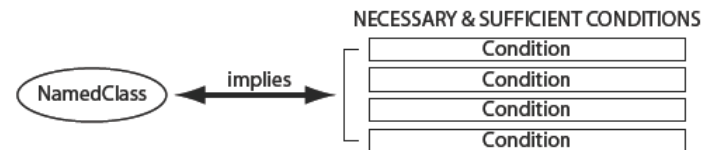
Necessary and Sufficient Conditions (II)

Protege 3.4

- ▼ Pizza
 - InterestingPizza
 - MeatyPizza
 - ▶ NamedPizza
 - NonVegetarianPizza
 - RealItalianPizza
 - SpicyPizza
 - SpicyPizzaEquivalent
 - VegetarianPizza
 - VegetarianPizzaEquivalent1
 - VegetarianPizzaEquivalent2
 - CheeseyPizza



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

Protege 3.4 interface showing the 'Asserted Conditions' panel. The panel displays the following conditions:

- Pizza
- hasTopping **some** CheeseTopping
- hasBase **some** PizzaBase

The 'Asserted Conditions' panel is circled in red. The 'NECESSARY & SUFFICIENT' condition is highlighted. The 'NECESSARY' and 'INHERITED' conditions are also visible, with the 'INHERITED' condition being [from Pizza].

Universal Restrictions (I)

All of the restrictions that we have created so far have been existential 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**.

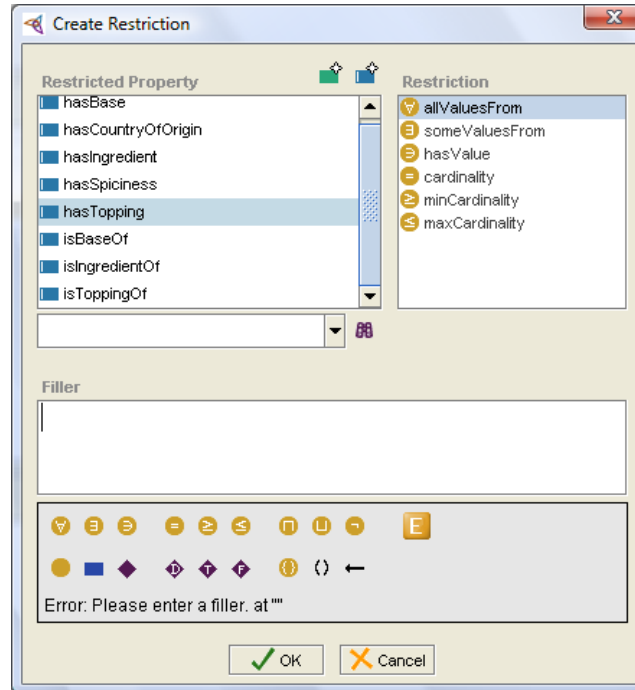
Universal restrictions constrain the relationships along a given property to individuals that are members of a specific class.

For example the universal restriction \forall 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



- ▼ Pizza
 - InterestingPizza
 - MeatyPizza
 - ▶ NamedPizza
 - NonVegetarianPizza
 - RealItalianPizza
 - SpicyPizza
 - SpicyPizzaEquivalent
 - VegetarianPizza
 - VegetarianPizzaEquivalent1
 - VegetarianPizzaEquivalent2
 - CheeseyPizza
 - VegetarianPizza2

Asserted Conditions

- NECESSARY & SUFFICIENT
- NECESSARY
- INHERITED

hasTopping only (VegetarianTopping or CheeseTopping)

hasBase some PizzaBase



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

The screenshot displays the Protege 3.4 ontology editor interface. On the left, a class hierarchy is shown with 'Pizza' as the root. Under 'Pizza', there are 'CheeseyPizza' and 'InterestingPizza'. 'NamedPizza' is a subclass of 'Pizza' and contains several subclasses: 'American', 'American_2', 'AmericanHot', 'Cajun', 'Capricciosa', 'Caprina', 'Fiorentina', 'FourSeasons', 'FruttiDiMare', 'Giardiniera', 'LaReine', 'Margherita', and 'Mushroom'. The 'Margherita' class is selected. On the right, the 'Asserted Conditions' panel for the 'Margherita' class is visible. It shows the following conditions:

- NECESSARY & SUFFICIENT**
 - NECESSARY**
 - NamedPizza
 - hasTopping **only** (MozzarellaTopping or TomatoTopping)
 - hasTopping **some** TomatoTopping
 - hasTopping **some** MozzarellaTopping
 - INHERITED**
 - hasBase **some** PizzaBase [from Pizza]

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**.

Protege 3.4

The screenshot shows the Protege 3.4 interface. On the left is a class hierarchy for 'Pizza' with subclasses: InterestingPizza, MeatyPizza, NamedPizza, NonVegetarianPizza, RealItalianPizza, SpicyPizza, SpicyPizzaEquivalent, VegetarianPizza, VegetarianPizzaEquivalent1, VegetarianPizzaEquivalent2, CheeseyPizza, VegetarianPizza2, and CustomizedPizza. The main workspace displays the 'Asserted Conditions' for the 'Pizza' class. A pink oval highlights the condition 'hasTopping min 3 PizzaTopping'. Below it is the condition 'hasBase some PizzaBase'. On the right, a panel shows the status of these conditions: 'NECESSARY & SUFFICIENT' for the highlighted condition, and 'NECESSARY', 'INHERITED', and '[from Pizza]' for the other.

Asserted Conditions

- NECESSARY & SUFFICIENT
- NECESSARY
- INHERITED
- [from Pizza]

hasTopping min 3 PizzaTopping

hasBase some PizzaBase

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

The screenshot shows the Protege 3.4 interface. On the left, a class hierarchy is displayed with 'NamedPizza' expanded, showing subclasses: American, FourCheese (highlighted), AmericanHot, Cajun, Capricciosa, Caprina, Fiorentina, FourSeasons, FruttiDiMare, Giardiniera, LaReine, and Margherita. The main window shows a table with columns 'Property', 'Value', and 'Lang'. Below the table, there are icons for adding, removing, and editing properties. The 'Asserted Conditions' panel on the right shows the following conditions:

Condition	Condition Type	Buttons
NamedPizza	NECESSARY & SUFFICIENT	[E]
hasTopping only (hasTopping exactly 4 CheeseTopping)	NECESSARY	[E]
hasBase some PizzaBase	INHERITED	[E]

The 'hasTopping' condition is highlighted in yellow, indicating it is the selected condition.

Datatype Properties

Creating a datatype property in the pizza example: **hasDiameter**.

The screenshot shows the Semantic Web Editor (SWE) interface for the 'pizza-ontology.owl' file. The 'Data Properties' tab is selected and highlighted with a pink circle. A 'Specify name' dialog box is open, prompting the user to enter a data property name. A blue line connects this dialog to a yellow box labeled 'Add Datatype Property'. The 'hasDiameter' property is listed in the 'Data Properties' panel. A blue line connects this list to a yellow box labeled 'Add Subproperty', which points to the 'Data Property Annotations' panel. The 'Data Property Annotations' panel shows 'Annotations: +'. The 'Data Property Usage' panel shows 'Data Property Annotations: hasDiameter'. The 'Data Property Description' panel shows 'Description: hasDiameter' and 'integer' as the range.

http://www.semanticweb.org/ontologies/2008/4/pizza-ontology.owl - [C:\Users\Man Carmen\ontologies\pizza-ontology\pizza-ontology.owl]

File Edit Reasoner Tools Refactor Tabs View Window Help

Active Ontology Entities Classes Object Properties **Data Properties** Individuals OWL Viz DL Query

Data Properties: hasDiameter

Delete selected properties

hasDiameter

Specify name

Please enter a data property name

Aceptar Cancelar

Add Datatype Property

Add Subproperty

Data Property Annotations

Data Property Usage

Data Property Annotations: hasDiameter

Annotations +

Data Property Description

Description: hasDiameter

Domains +

Ranges +

integer

Equivalent properties +

Super properties +

Disjoint properties +

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.

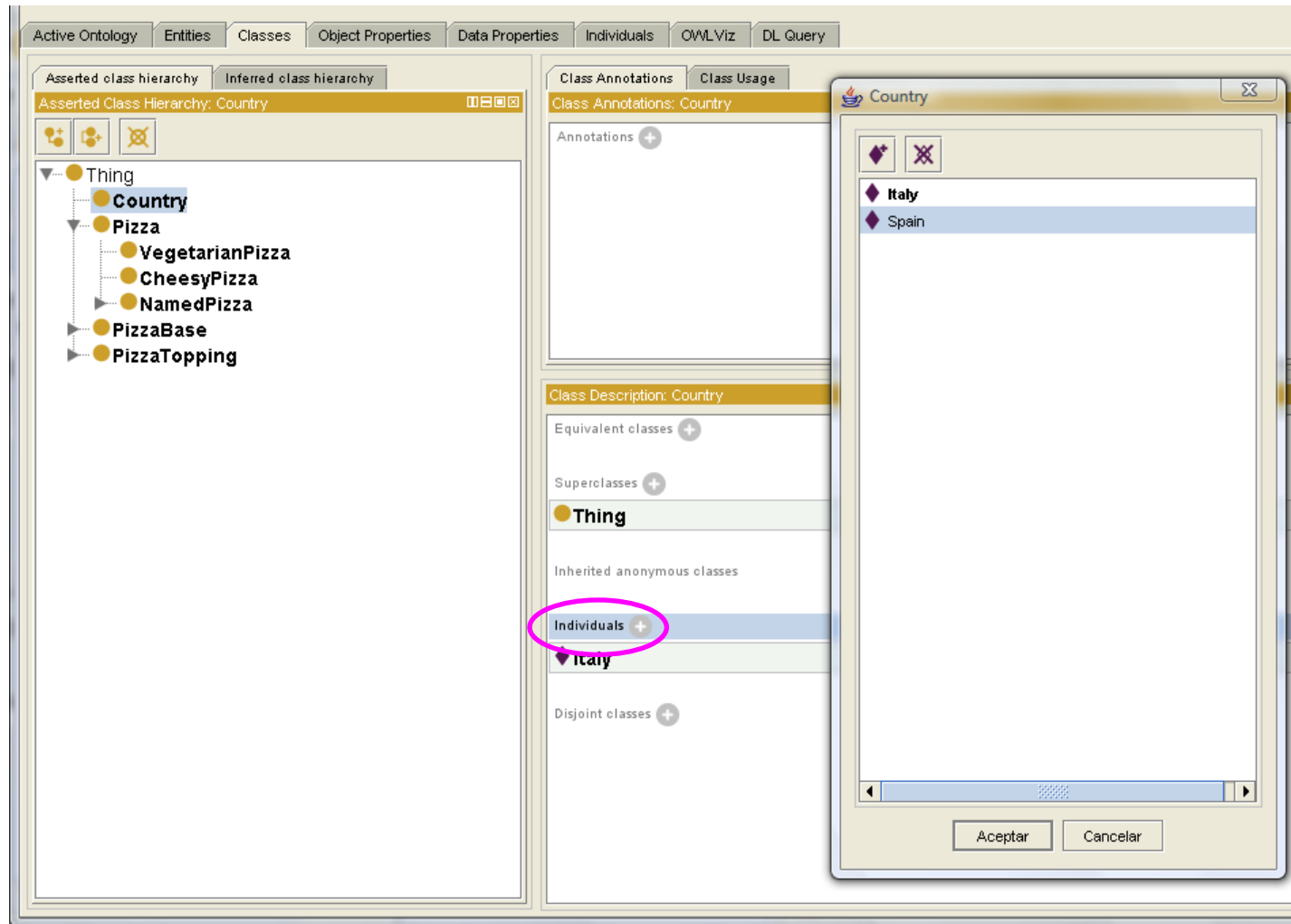
Create a Vegetarian Pizza, which have no meat and no fish toppings.

Create a Real Italian Pizza, which only have bases that are ThinandCrispy.

Create a subclass of Named Pizza with a topping of Mozzarella.

Individuals

Creating **individuals** of class Country.



hasValue Restriction

Specifying **Italy** as country of origin for Mozzarella.

The screenshot displays the Protégé OWL editor interface. On the left, the 'SUBCLASS EXPLORER' shows the 'Asserted Hierarchy' for the project 'pizza.owl'. The hierarchy includes 'owl:Thing' as the root, followed by 'DomainConcept', 'Country', 'IceCream', 'Pizza', 'PizzaBase', and 'PizzaTopping'. Under 'PizzaTopping', there is a 'CheeseTopping' subcategory, which includes 'FourCheesesTopping', 'GoatsCheeseTopping', 'GorgonzolaTopping', 'MozzarellaTopping', and 'ParmesanTopping'. 'MozzarellaTopping' is currently selected.

In the center, the 'CLASS EDITOR' is open for the 'MozzarellaTopping' class. It shows a list of properties: 'rdfs:comment', 'rdfs:label', 'hasCountryOfOrigin', and 'hasSpiciness'. The 'hasCountryOfOrigin' property is highlighted with a pink oval, and its value is set to 'Italy'.

On the right, the 'Create Restriction' dialog is open. It shows the 'Restricted Property' list with 'hasCountryOfOrigin' selected. The 'Restriction' list on the right includes 'allValuesFrom', 'someValuesFrom', 'hasValue', 'cardinality', 'minCardinality', and 'maxCardinality'. The 'hasValue' restriction is selected. The 'Filler' field contains the text 'Italy'. At the bottom of the dialog, there are 'OK' and 'Cancel' buttons.