Week 7 Graded Assignment Guide

A Practical Introduction to Ontologies and OWL

This assignment guide contains an introductory-level tutorial that is aimed at newcomers to ontologies and those using formalisms other than OWL. It will provide you with hands-on experience in the construction of ontologies as you follow the instructions in the tutorial and build an OWL ontology using the domain of Pizzas.

Credit: The University of Manchester, with Protégé-4 diagrams and updates added by P. Browne and adapted for Protégé-5.5.0 by J. Lee.

Context

Ontologies are a critical piece of the Semantic Web jigsaw puzzle, and are already used in various forms to capture knowledge in a machine understandable language. The Web Ontology Language (OWL) is a W3C standard that allows the formalization of knowledge in a semantically rich model with explicit meaning.

Goals of tutorial

This introductory level tutorial is aimed at newcomers to ontologies and those using formalisms other than OWL and will provide attendees with hands-on experience in the construction of ontologies.

Participants will build an OWL ontology using the domain of Pizzas.

Learning Objectives

Students will:

- Understand some of the OWL language elements, and their explicit semantics
- Learn the basic principles of modeling using Description Logic based ontologies
- Use informal modeling techniques to uncover issues relating to knowledge capture
- Gain hands-on experience with ontology development using the Protégé-OWL tools
- Learn how to take advantage of inferencing capabilities to build robust, reusable models
- Learn how to query an ontology.
- Learn how to create instances of classes.

The Pizza Tutorial

The following tutorial is based on pizza ontology courses that have been run at The University of Manchester for a number of years. Pizzas have been chosen as they are fun, well-known, fairly neutral (although we wildly encourage arguing about whether the model is correct), have a relatively small scope and are highly compositional. We are building an ontology to support a menu querying system (something along the lines of the PizzaFinder application (www.co-ode.org/downloads/pizzafinder/) . While building your model, try to keep the application in mind.

Please do a *Save* regularly to version your work – it is easy to make mistakes while modeling that are really difficult to track down - we do not want you to lose work.

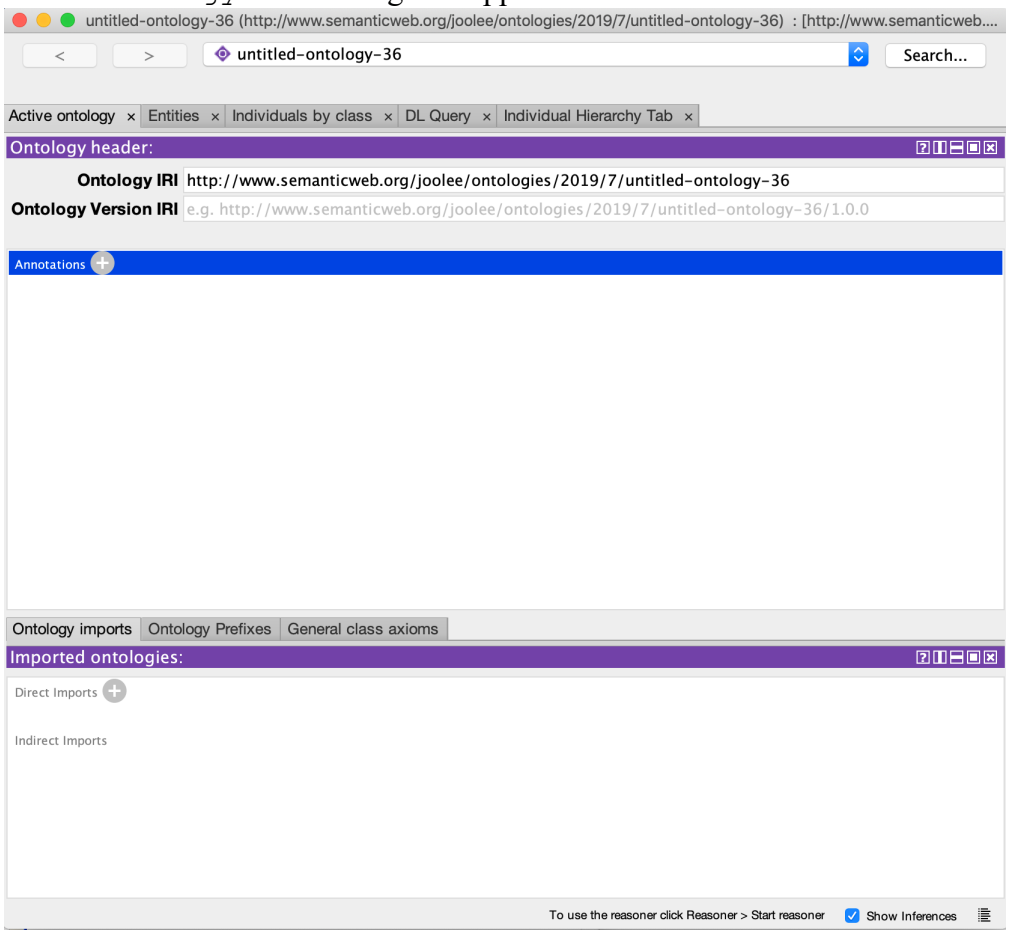
Part 1: Introduction to Protégé and Primitive Classes in OWL

Starting Protégé and Creating a new project

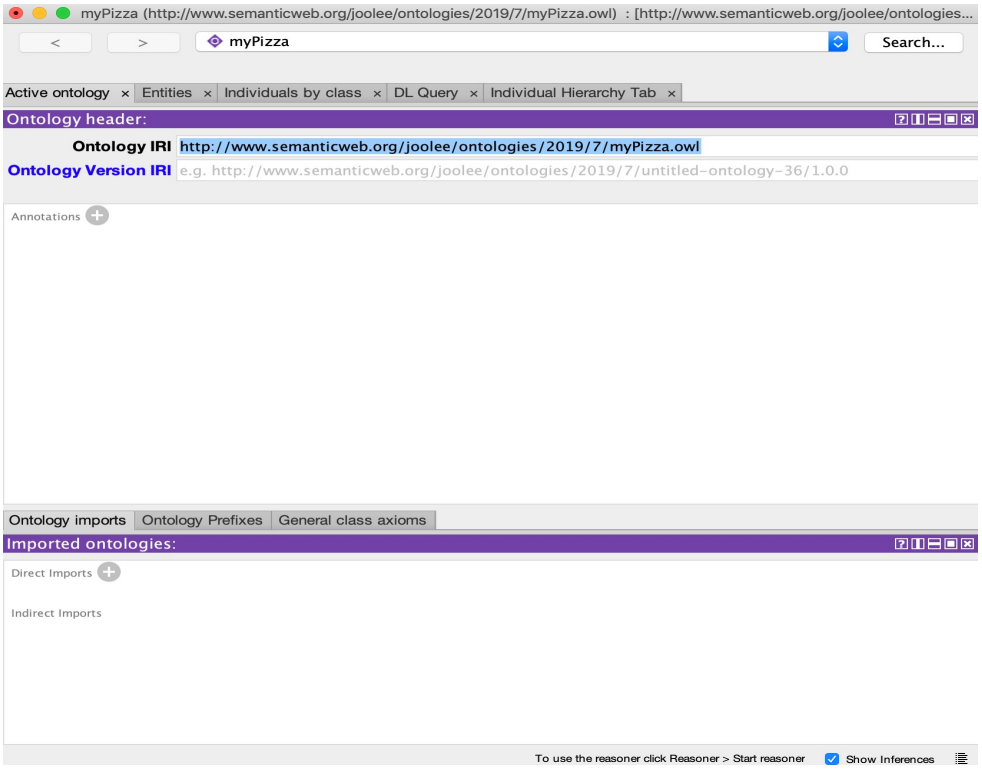
1. Please start Protégé if you haven't already – it should be available in your Application or Program Files folder if you have installed it correctly. You will be presented with a startup dialog.

2. Select File -> New…

3. The `Ontology URI` dialog will appear:



4. Edit the final part of the URI to `myPizza.owl`.

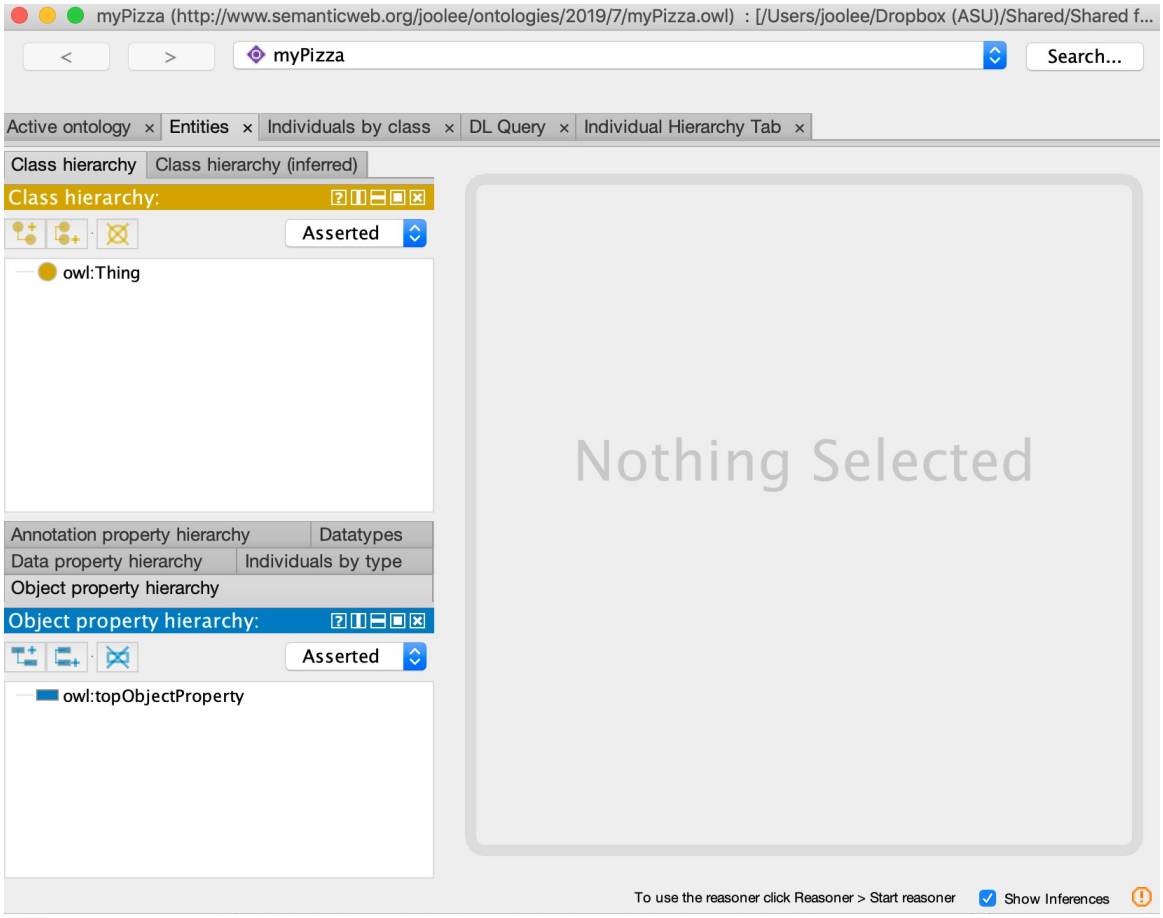


5. Select File -> Save. You have an option to save. The default is "RDF/XML Syntax". It will prompt you to a folder to save the file. Save as "myPizza" and click the save button. You may want to save the file time to time when you make updates.

Exercise 1: Create a Primitive Class Hierarchy

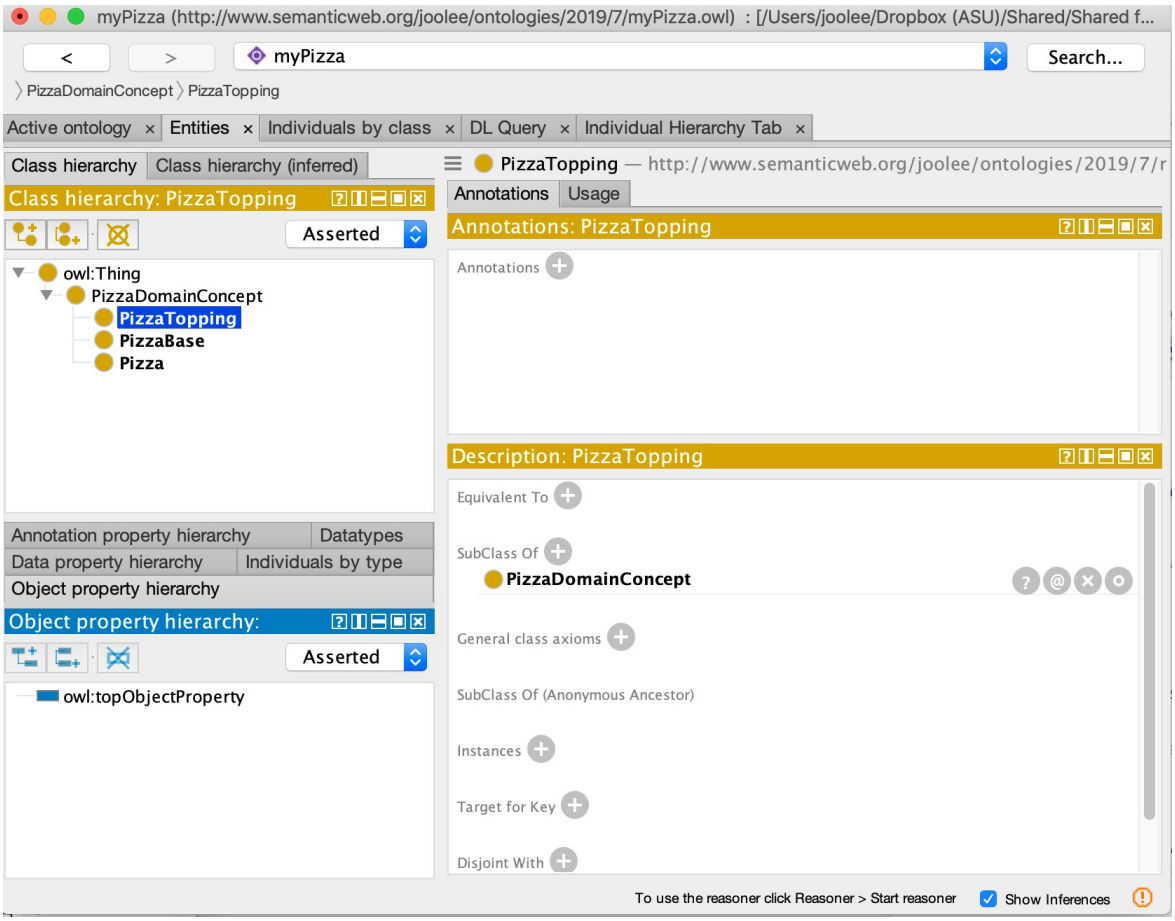
You will now begin to create the primitive classes in your model. This begins with a couple of top-level concepts, including somewhere to put your toppings from the ingredient cards.

1 Select `Entities` tab and then `Class hierarchy` tab.



2 Choose "**`owl:Thing`**" and create the following subtree using the `Create Subclass` and `Create Sibling Class` buttons in the `class`

`hierarchy` tab.



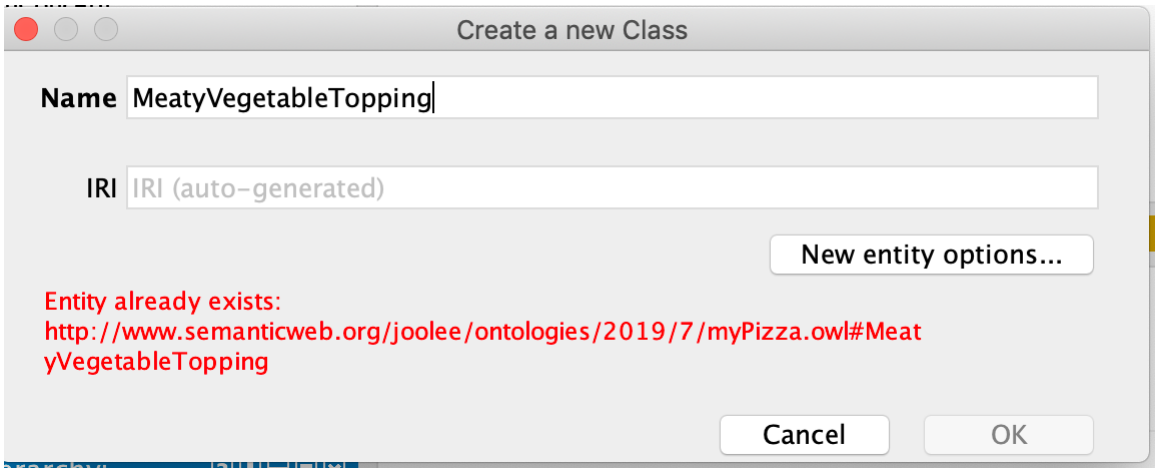
3 Create further subclasses of your pizza toppings under **`PizzaTopping`**, using the ... **`Topping`** for each topping. Make sure you have at least included the following as they will be used later:

- **`MeatTopping`**
- **`CheeseTopping`**
- **`MozzarellaTopping`**
- **`TomatoTopping`**
- **`VegetableTopping`**

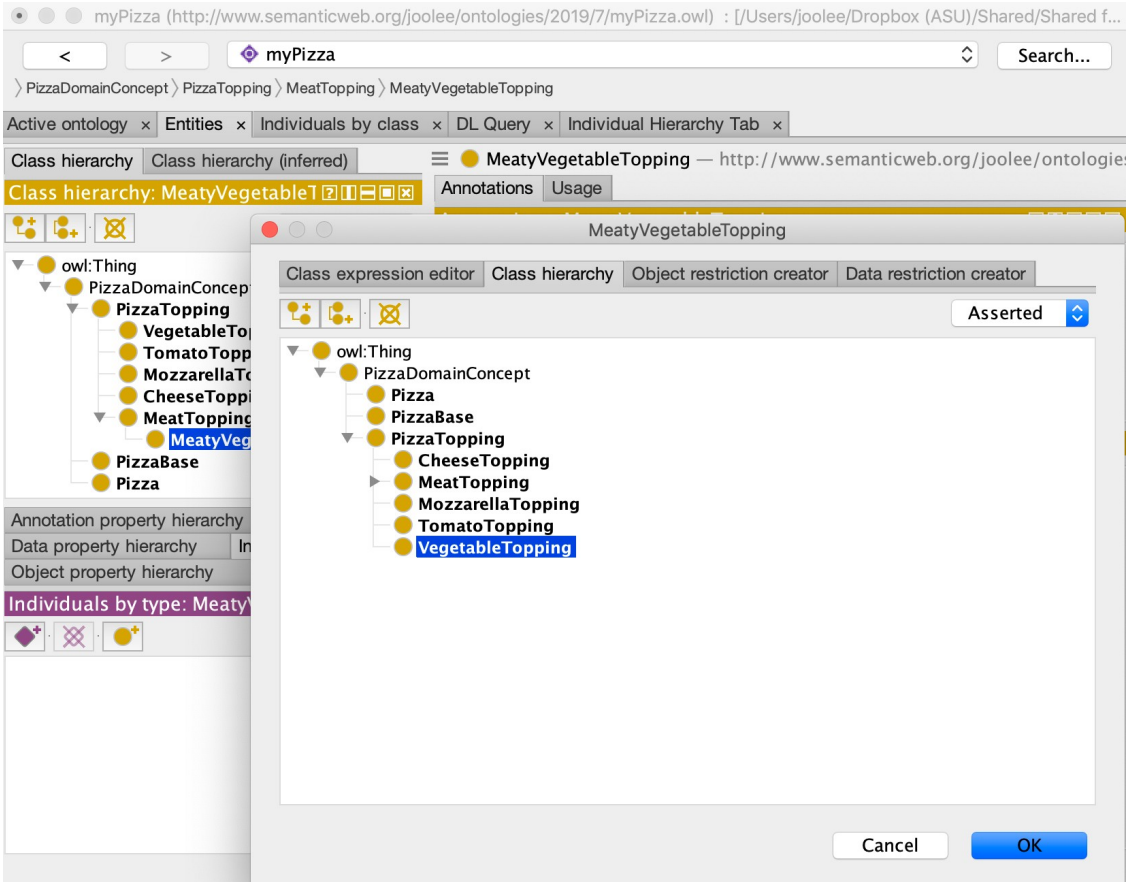
Notice that we are adding a suffix of **Topping** to all the classes – this is to allow for the later possibility of extending our ontology to talk about other types of food such as salads, where Tomato might be different from "Tomato pizza topping"

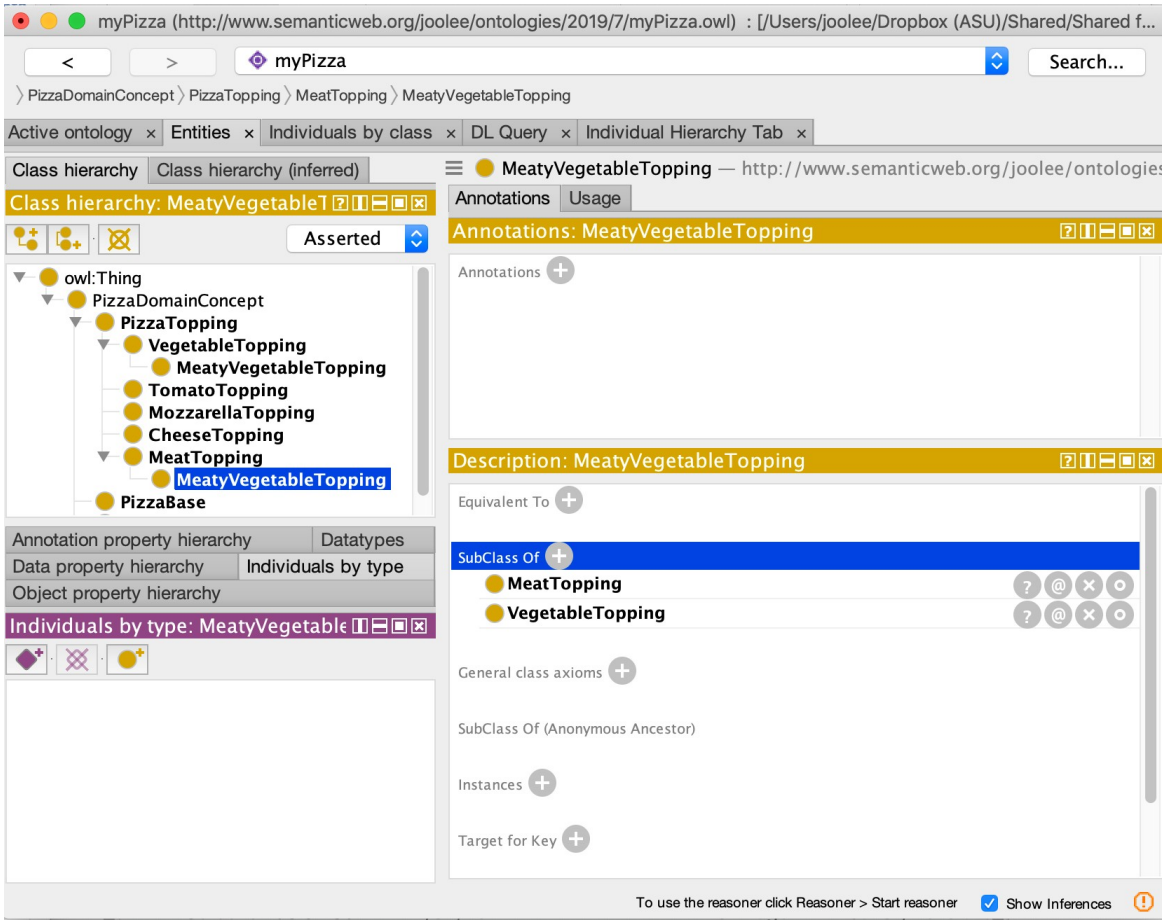
4 We cannot have two classes with same name to the class hierarchy using `Add subclass` or `Add Sibling class` buttons. However, using the techniques in Step 6, we can create classes with multiple parents. These classes will appear in several places in the class hierarchy.

5 Create a **MeatyVegetableTopping** class as a subclass of **MeatTopping** and **VegetableTopping.** You should get an error when you try to make the second version of **MeatyVegetableTopping**.



6 To allow a child to have multiple parents, we will make one version of **MeatyVegetableTopping** under, say, **MeatTopping** and then make **VegetableTopping** a second parent class of the same **MeatyVegetableTopping**. We do this by selecting MeatyVegetableTopping in `Class hierarchy` tab and click cross ⊕ of `Subclass Of` button. It will open `Class hierarchy` view. . Select **VegetableTopping** as the new additional parent of **MeatTopping.** This way of working can be summarized as follows 1) select the child that you want to have multiple parents, 2) click on `SubClass Of` ⊕ in the `Class description` view, 3) navigate to the new parent class in `Class hierarchy` tab as below


myPizza (http://www.semanticweb.org/joolee/ontologies/2019/7/myPizza.owl) : [/Users/joolee/Dropbox (ASU)/Shared/Shared f...
myPizza
Search...
PizzaDomainConcept
PizzaTopping
MeatTopping
MeatyVegetableTopping
Active ontology
Entities
Individuals by class
DL Query
Individual Hierarchy Tab
Class hierarchy
Class hierarchy (inferred)
Class hierarchy: MeatyVegetableT
MeatyVegetableTopping — http://www.semanticweb.org/joolee/ontologie
Annotations
Usage
owl:Thing
PizzaDomainConcep
PizzaTopping
VegetableTo
TomatoTopp
MozzarellaTo
CheeseToppi
MeatTopping
MeatyVeg
PizzaBase
Pizza
Annotation property hierarchy
Data property hierarchy
Object property hierarchy
Individuals by type: Meaty
MeatyVegetableTopping
Class expression editor
Class hierarchy
Object restriction creator
Data restriction creator
Asserted
owl:Thing
PizzaDomainConcept
Pizza
PizzaBase
PizzaTopping
CheeseTopping
MeatTopping
MozzarellaTopping
TomatoTopping
VegetableTopping
Cancel
OK

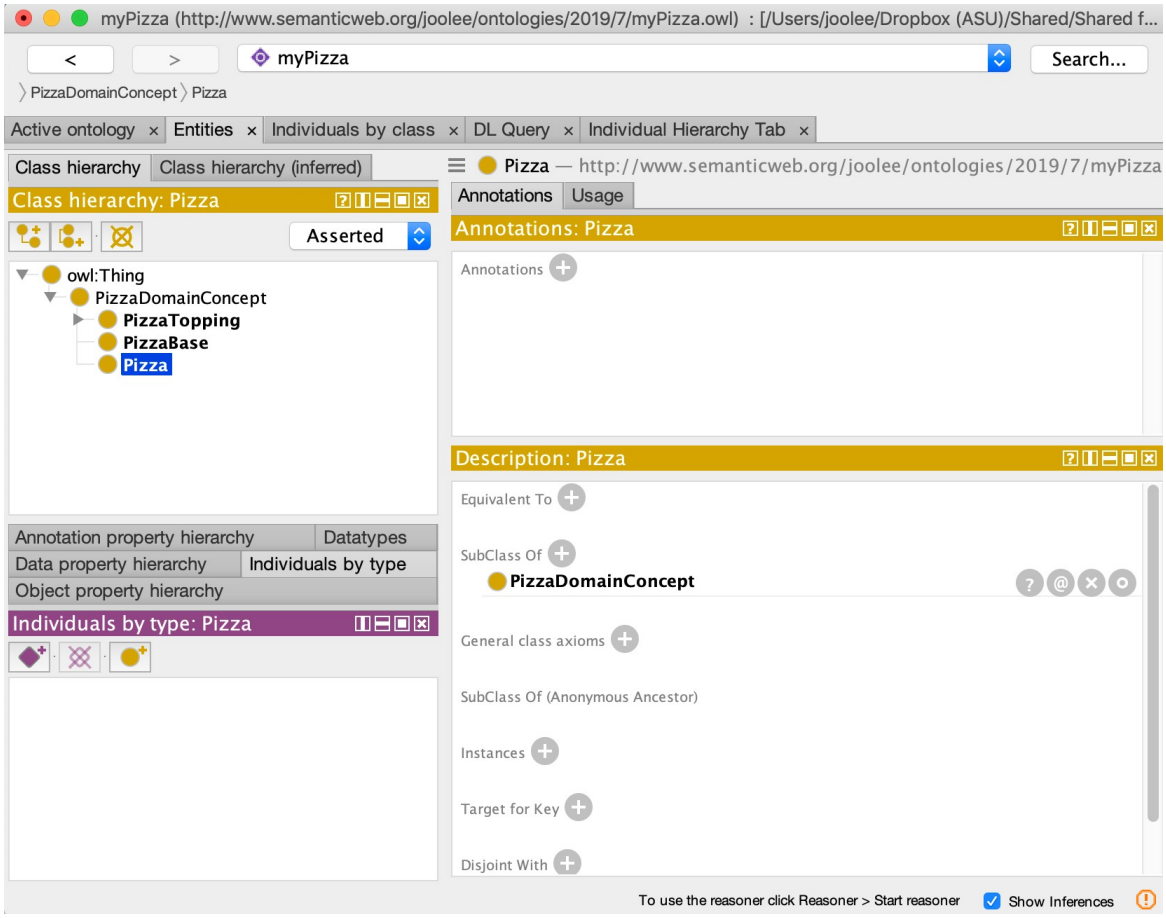


You should now have a small hierarchy of **`PizzaTopping`**s. Add to these whenever you have a few minutes as they will allow you to create a larger ontology of pizzas later on.

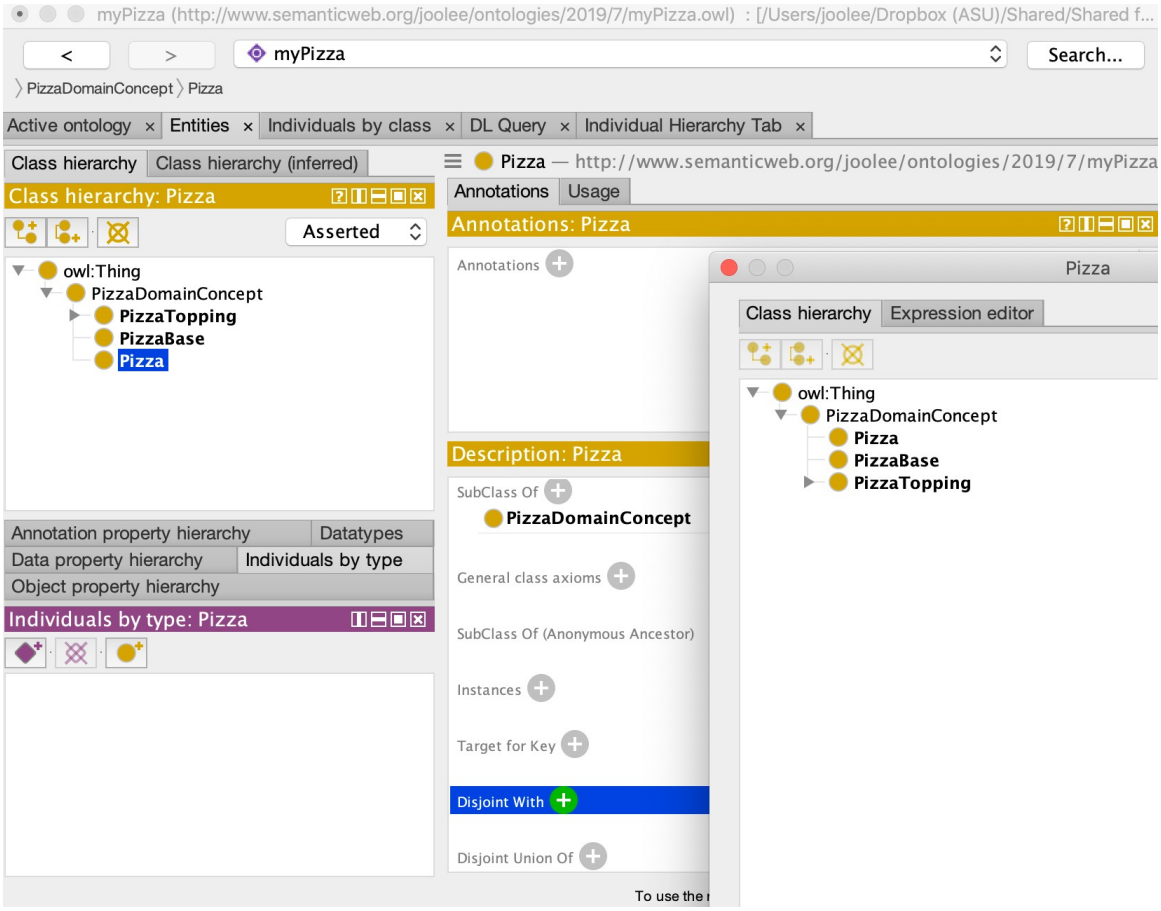
Exercise 2: Add disjoints

To make sure toppings cannot be both meat and vegetable at the same time, add disjoints in to your primitive tree. You will need to run a *reasoner* to check the consistency of your ontology. Click the reasoner menu and you'll see the list of the reasoners being supported in Protégé. Select Fact++ and run `Start Reasoner`. This will create an inferred class hierarchy. We will use this technique to check a set of Disjoint Classes. We first make some disjoint classes as follows.

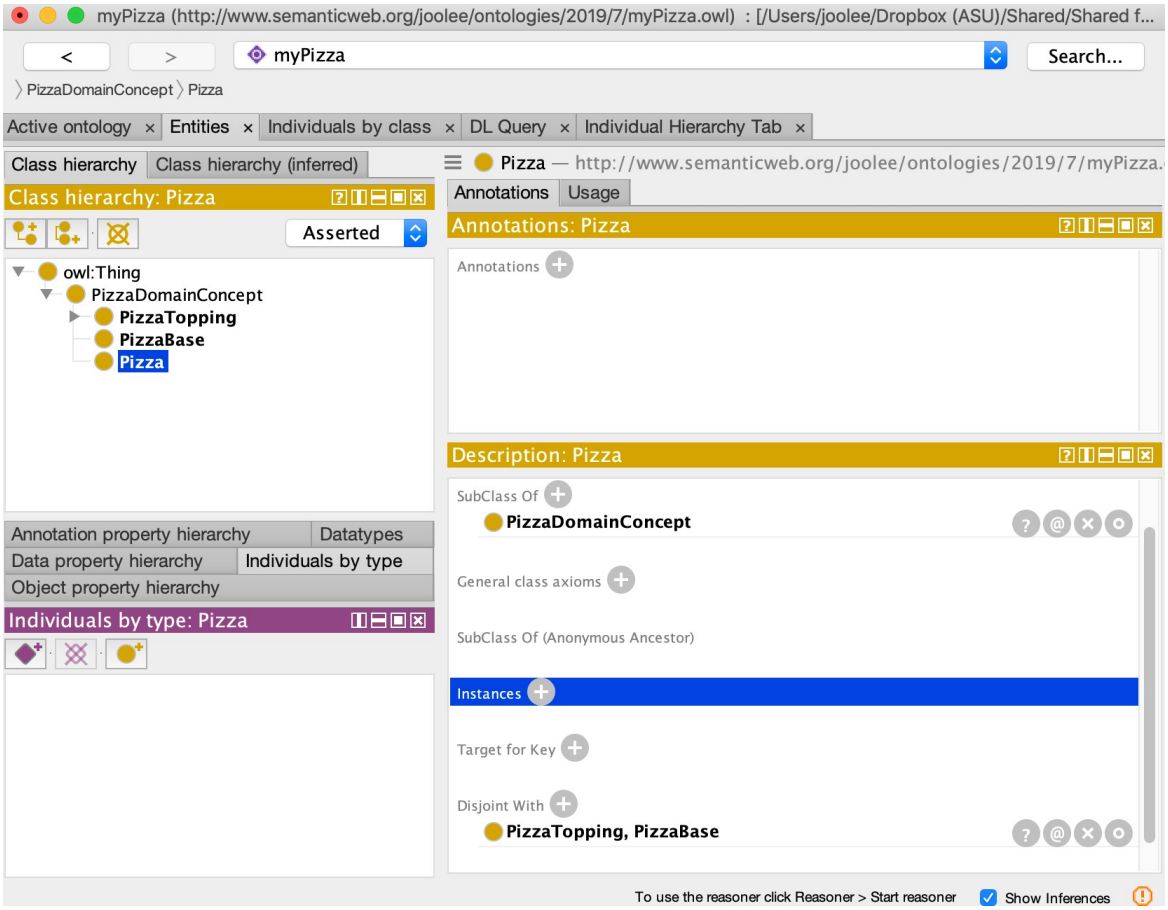
1. Select one of your top level concepts (eg **Pizza**)



2. Select `Disjoint with` from the `Description` view, which will open the window below (called Pizza):



Hold down the Shift (or Ctrl) key and Select **PizzaBase** and **PizzaTopping**. Click OK This will select multiple classes as being disjoint from **Pizza**. You should see the following screen. Note the Pizza `Class description view` has two disjoint classes under the `Disjoint classes button`.



3. From the main tool bar select `Start reasoner`. Once your reasoner is running, you can reclassify your ontology by selecting `Synchronize reasoner` from the menu.

4. Using the technique in Step 2, make all the sublacasses of **`PizzaTopping`** disjoint.

5. Run `Synchronize reasoner`.

Q1. Do any of your classes come out as inconsistent? (They will be shown in red in the `Class hierarchy` tab; you may need to expand to see the red.) Explain why and describe a way to resolve the inconsistency. [NOTE: Save your answer to this question, as you will need it to complete the assignment in Coursera.]

Exercise 3: Properties and Restrictions

In order to describe our classes we need properties, which are used to relate members of a class. We then add restrictions on the class to state logically how these properties are used. In general, we can have:

- Quantifier Restrictions:

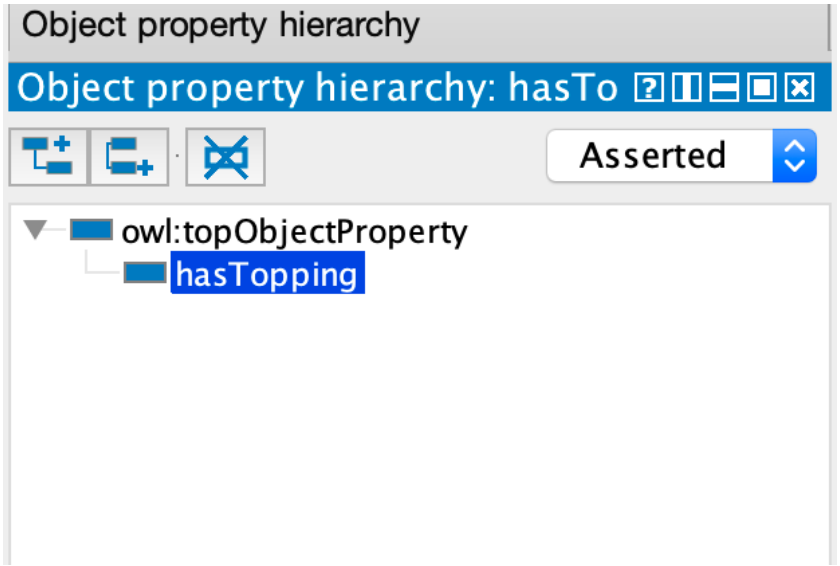
    - Existential (pizzas that have vegetable toppings)
    - Universal (pizzas that only have vegetable toppings)
- Cardinality Restrictions
    - Pizzas have exactly one base
- `hasValue` Restrictions
    - A `hasValue` restriction, denoted by the symbol ∋, describes the set of individuals that have at least one relationship along a specified property to a specific individual. Pizzas that have Italy as their country of origin (similar to existential).

At this stage we are creating *Primitive Classes,* which only have *Necessary Conditions* on them – these are conditions that must be satisfied by all members of this class. The following steps assert the two restrictions:

`MargheritaPizza hasTopping MozzarellaTopping`
`MargheritaPizza hasTopping TomatoTopping`

Step 1. Create Properties

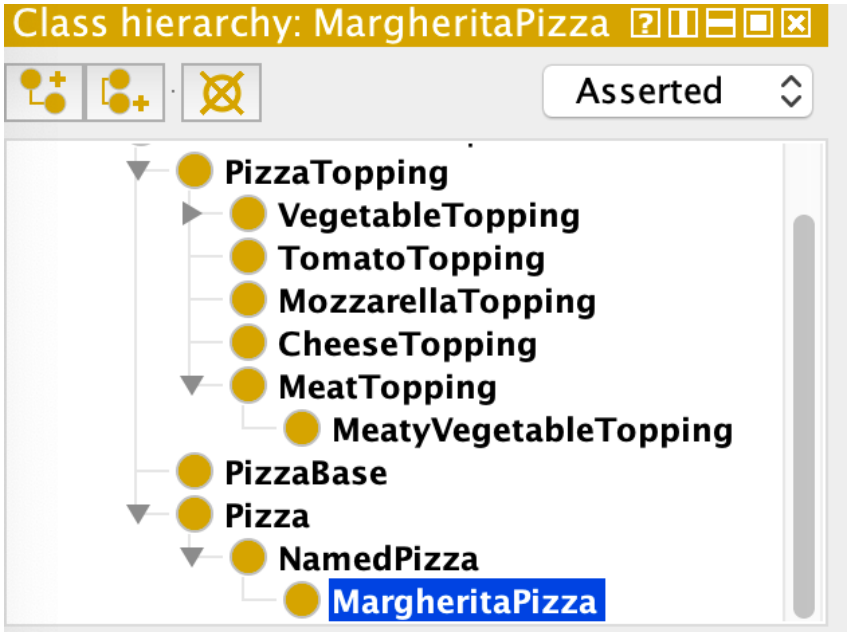
1. Select the `Object Property hiearchy Tab.` It already contains owl:**topObjectProperty.**
2. Create a new Object Property, **`hasTopping,`** by using the leftmost button.



Step 2. Create a Named Pizza

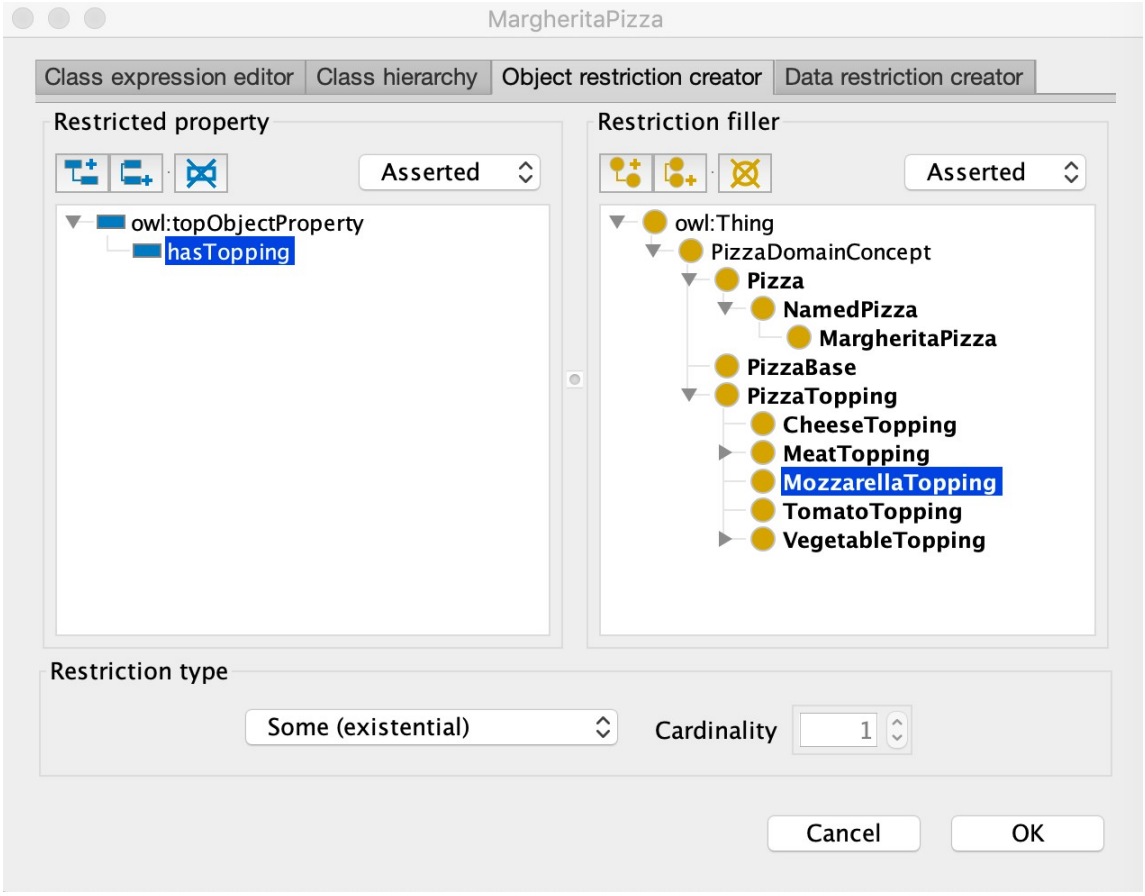
1 In the `Class Hierarchy` tab, create a new subclass of **`Pizza`** called **`NamedPizza`**

2 Create a subclass of **`NamedPizza`** called **`MargheritaPizza.`** `The hierarchy should be as follows:`

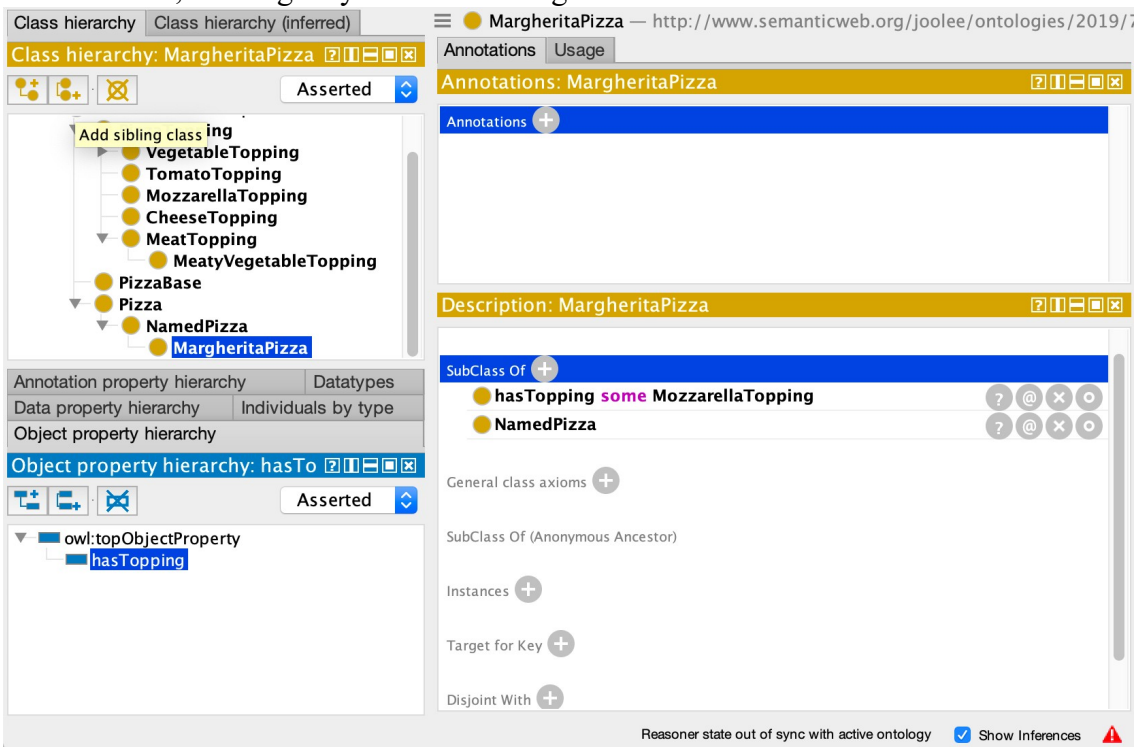


Step 3. Create restrictions on `MargheritaPizza`

1. Select the `Class Hierarchy` tab
2. Select **`MargheritaPizza`**
3. Select `subclass Of` ⊕ from the `Class Description` view, and select Object restriction creator.
4. Create a `Some(Existential)` restriction along the **`hasTopping`** property for **`MargheritaPizza`**, which relates it to **`MozzarellaTopping`**. Navigate to **`MozzarellaTopping`**.



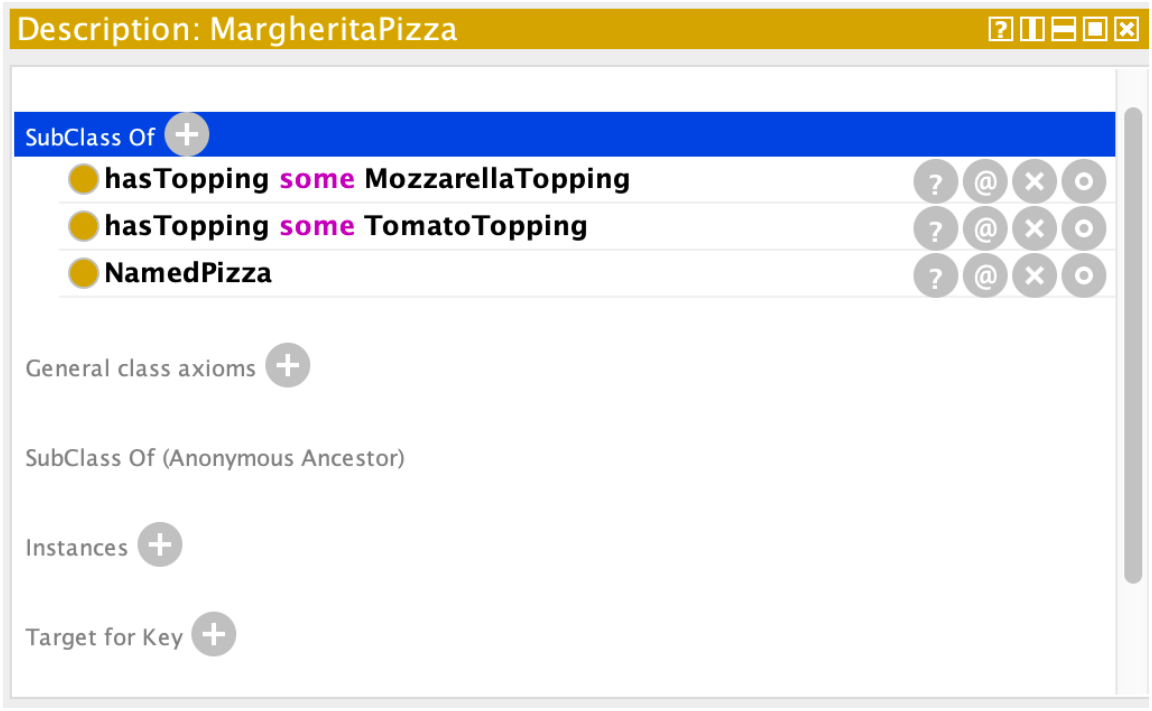
5. Click OK, which give you the following:

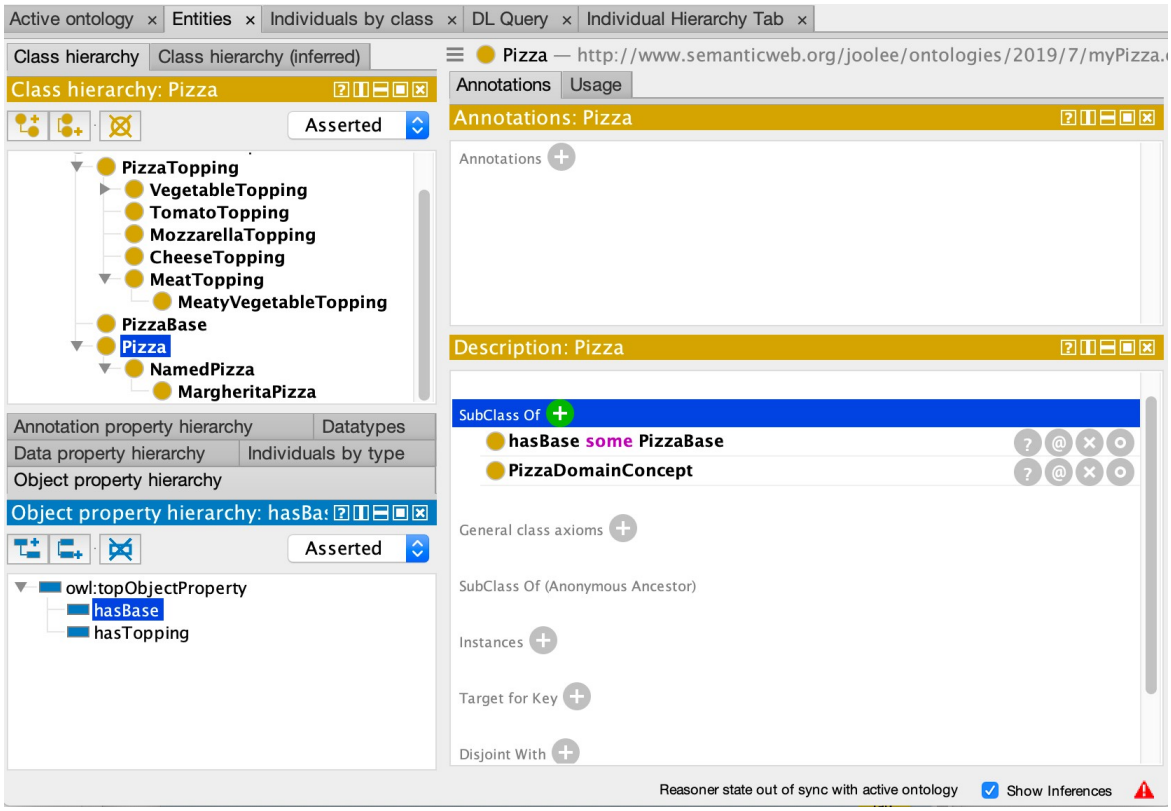


Which says "*A margherita pizza has at least one mozzarella topping*". Note that by default, restrictions are created as *Necessary Conditions* unless the *Necessary & Sufficient* heading is selected – for creating *primitive classes*, only create Necessary Conditions.

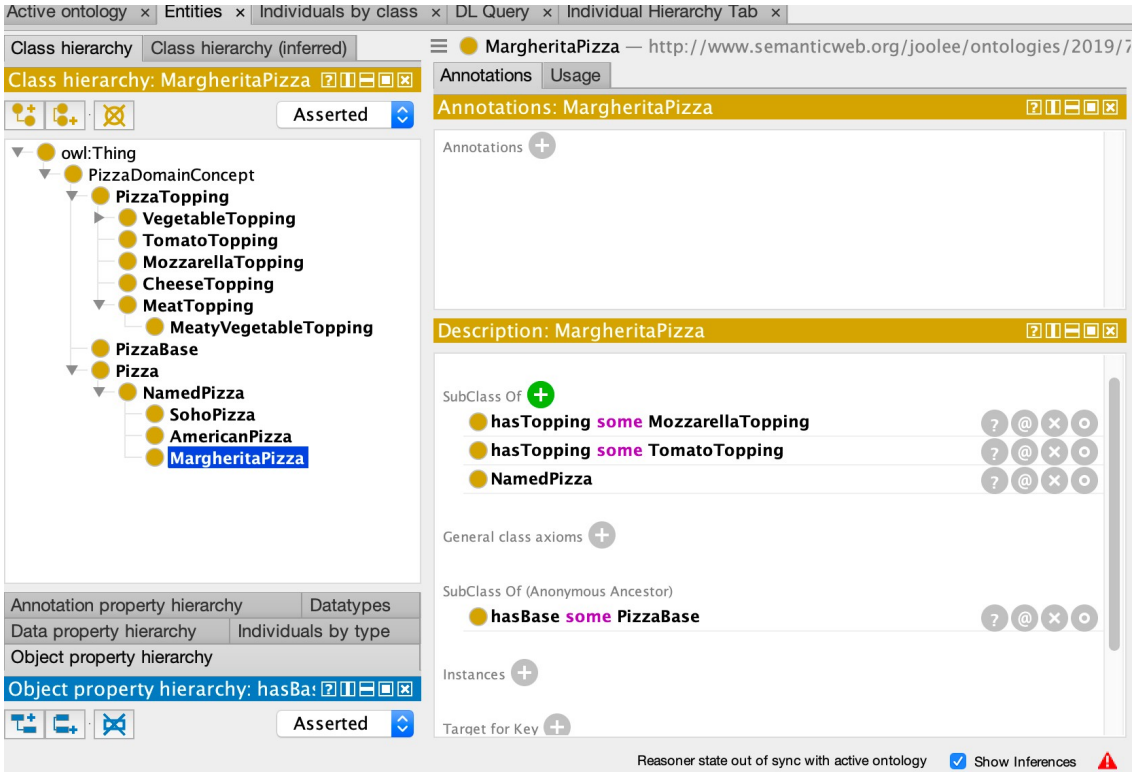
6. Add a further existential restriction on **`MargheritaPizza`** to state that it must have at least one topping from **`TomatoTopping .`**

7. Create a restriction on **`Pizza`** to state that:
"*All pizzas must have at least one base from PizzaBase*" You will first need to create the property **`hasBase`**. The result should be:





8. Create some other pizzas and add toppings (ingredients) in the same fashion – one restriction per topping. Make sure you create at least one pizza that has some meat on it. Such as:



An **AmericanPizza** is similar to a **MargheritaPizza** but with an extra topping of pepperoni (**PepperoniTopping**), which is a meat. A **SohoPizza** is similar to a **MargheritaPizza** but has additional toppings of olives (**OliveTopping**) and parmezan (sic) cheese (**ParmezanTopping**).
Note that all subclasses of **NamedPizza** inherit the properties of **Pizza**. Which, at the moment, means that they have at least one base.

Part 2: Defined Classes and Additional Modelling Constructs in OWL

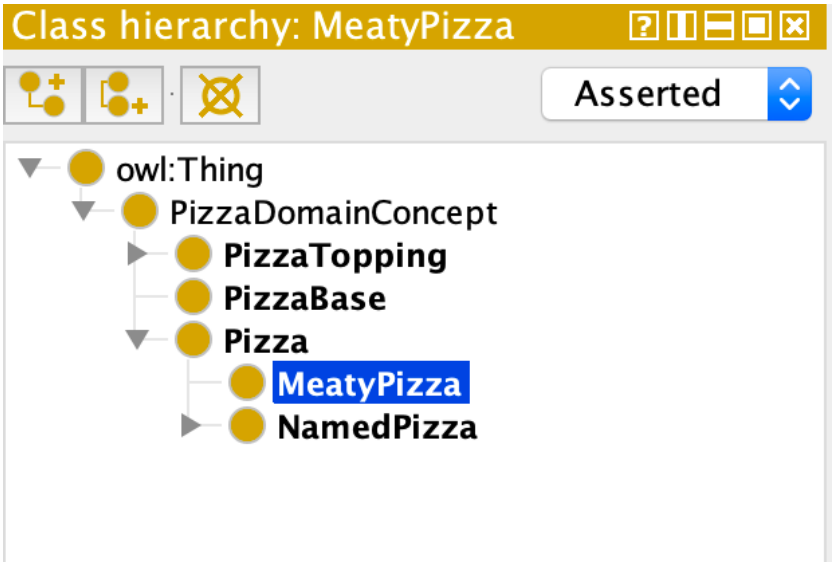
Exercise 4: Define a `MeatyPizza`

Creating a defined class is similar to creating a primitive class, but a defined class has one or more *Necessary & Sufficient* Conditions. Classes can easily be migrated between primitive and defined. Using a defined class we can indentify the class of an object. In English we might say:
"*I know it's a Cheesy Pizza <u>because</u> it has cheese on it*".
Note that this is different from the rule "*A Cheesy Pizza must have cheese on it*"
In a Protégé (OWL) class hierarchy we say subsumption means <u>necessary</u> implication
With Protégé we can use subsumption reasoning where an asserted instance A (sub-class) related to B (super-class) through the subtype tree. We can infer the more general class. Subclasses are subsumed by their superclasses (e.g. All cats are animals). In relation to our hierarchy this means that "All PepperoniTopping is MeatTopping", that "All MeatTopping is a PizzaTopping", etc. Using Protégé it is possible to automatically compute a classification hierarchy and check for inconsistencies.

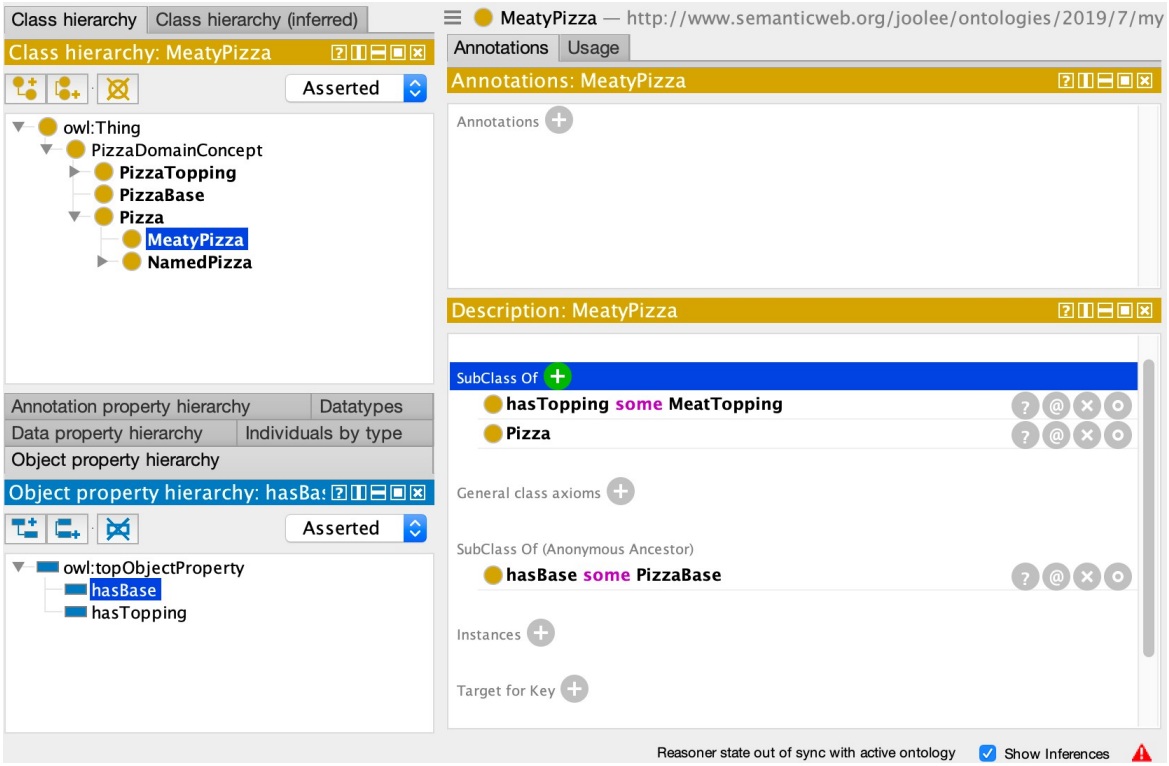
Say we wish to model the statement:

*Any **Pizza** that has at least one topping from **MeatTopping** is a **MeatyPizza***
or
A ***MeatyPizza*** is any pizza which, *amongst other things*, has only toppings that are ***MeatTopping***.

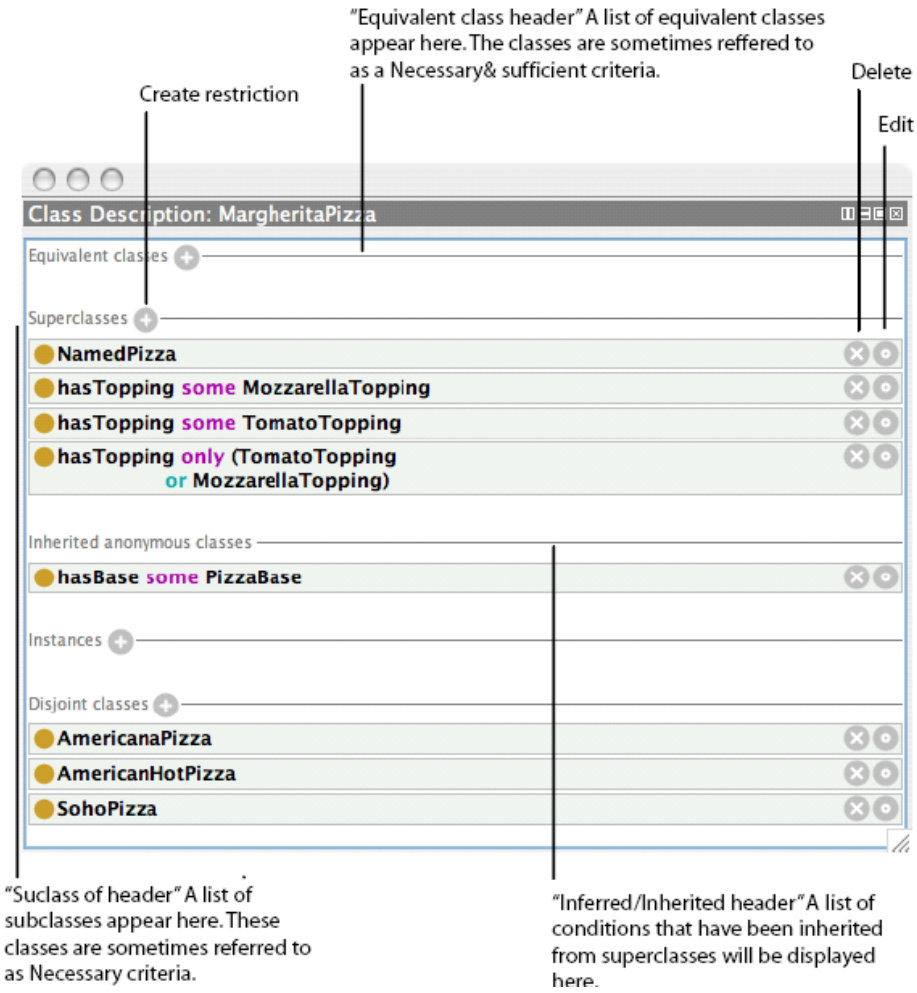
1. Create a new subclass of **Pizza** called **MeatyPizza.** In general, defined classes are **not** disjoint from their siblings, giving:



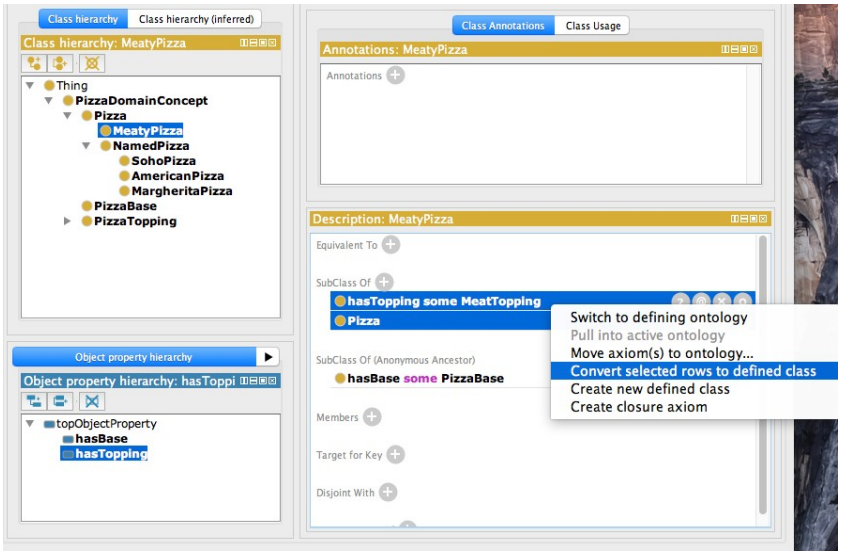
2. Create a restriction on **`MeatyPizza`** to state that it has at least one topping from **`MeatTopping`**, giving.



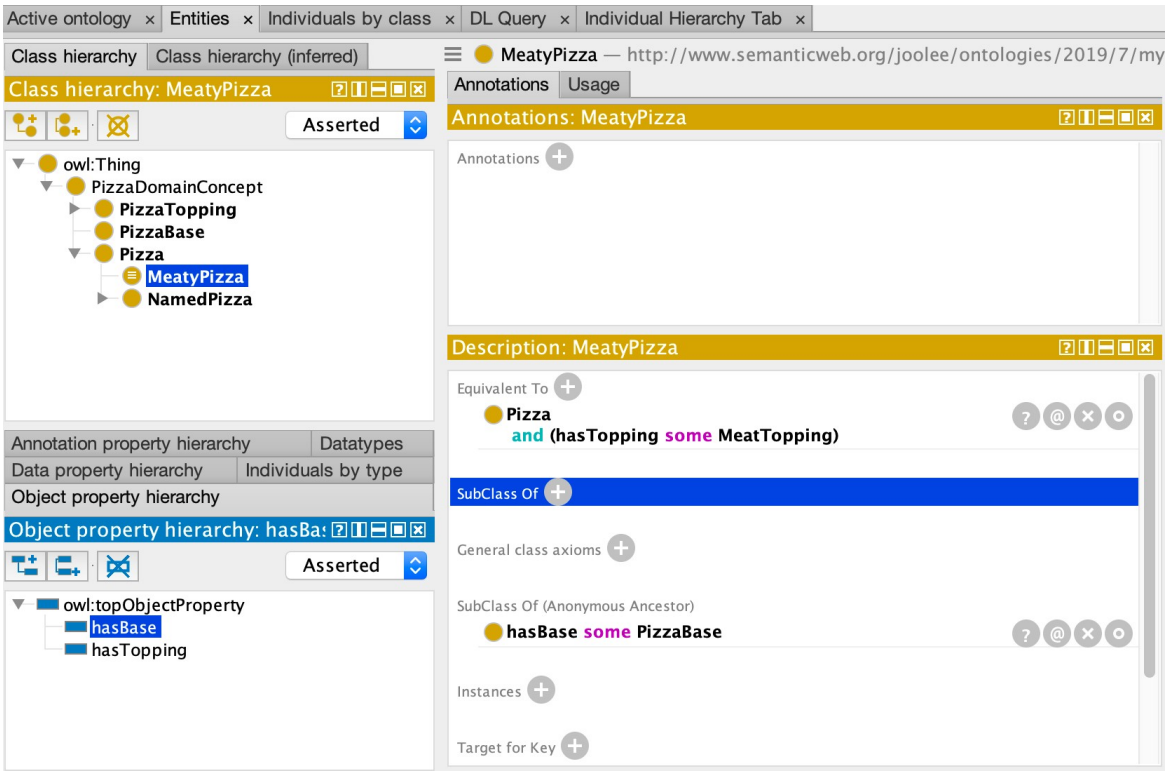
3. The `Equivalent To` section of the `Class description view` contains a list of equivalent classes, which are sometimes referred to as a Necessary & Sufficient criteria. Please study the following screen.



4. We are now going convert the <u>necessary</u> condition for **`MeatyPizza`** into <u>necessary & sufficient</u> conditions. In the `Class description view` for **`MeatyPizza`** we will move both conditions (the new restriction and the superclass, **`Pizza)`** up to the `Equivalent Classes` section. To do this highlight both `Pizza` and `hasTopping some MeatTopping` as below by selecting with Shift key on. Then right-click and select `Convert selected rows to defined class`.



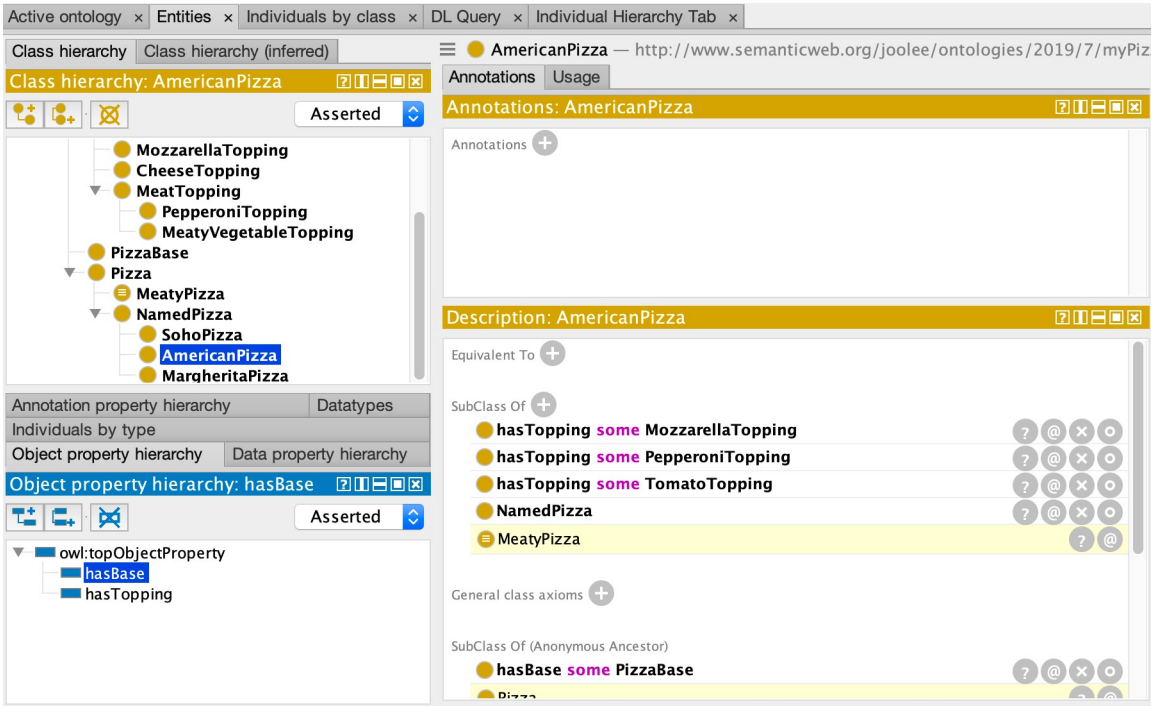
If you click on the **`MeatyPizza`** icon it should change to an *equivalent* icon .
The result should be as below:



The above screen asserts:

```
MeatyPizza = Pizza and (hasTopping some MeatTopping)
```

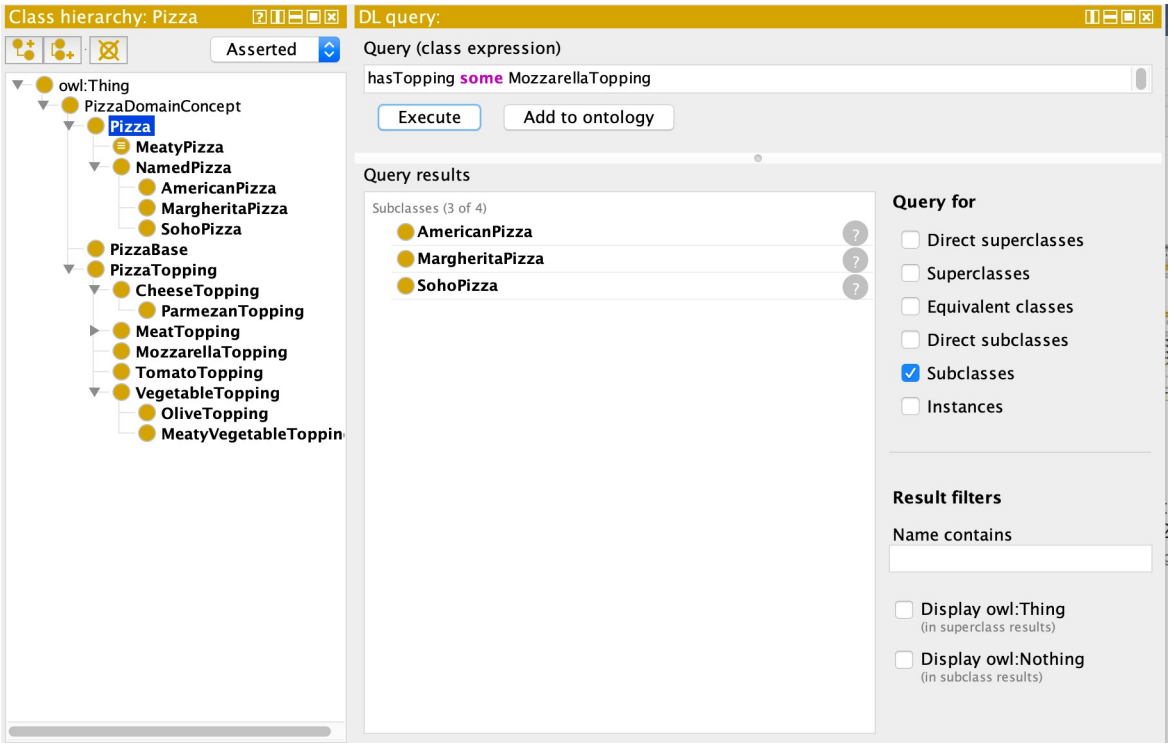
5. Classify your ontology and check the Inferred Hierarchy. Select "Class hierarchy (inferred)" tab. You should find that the reasoner correctly inferred that an **`AmericanPizza`** is a **`MeatyPizza`**, as below.



Part 3: Querying and creating individuals

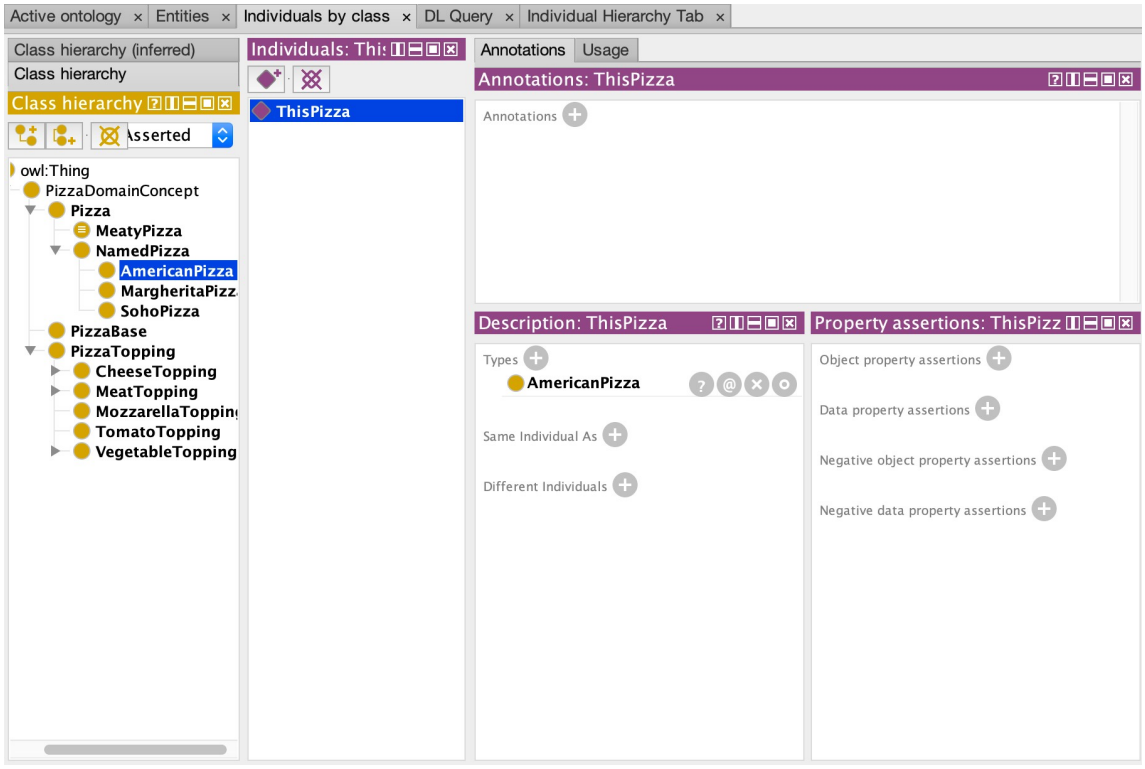
1 In the DL Query tab, type the following queries, pressing *`Execute`* each to see each result.

```
Pizza
hasTopping some MozzarellaTopping
```

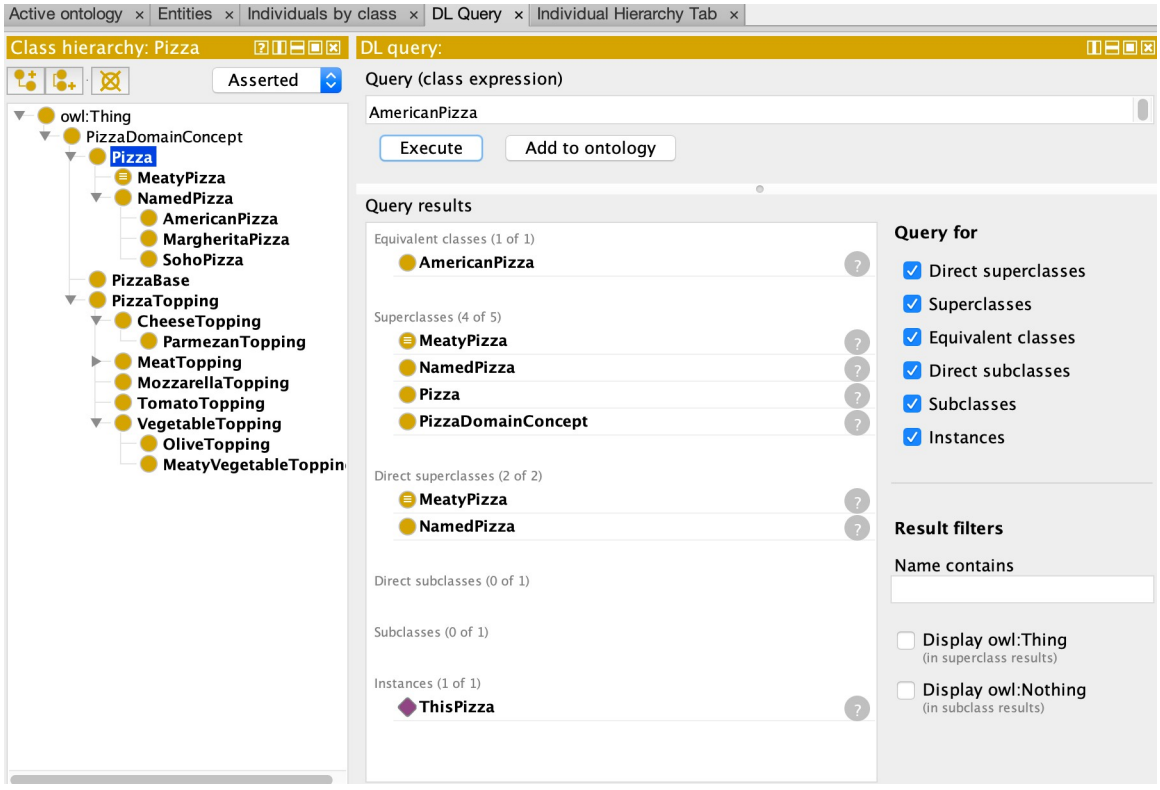


For more examples see
http://protegewiki.stanford.edu/index.php/DLQueryTab

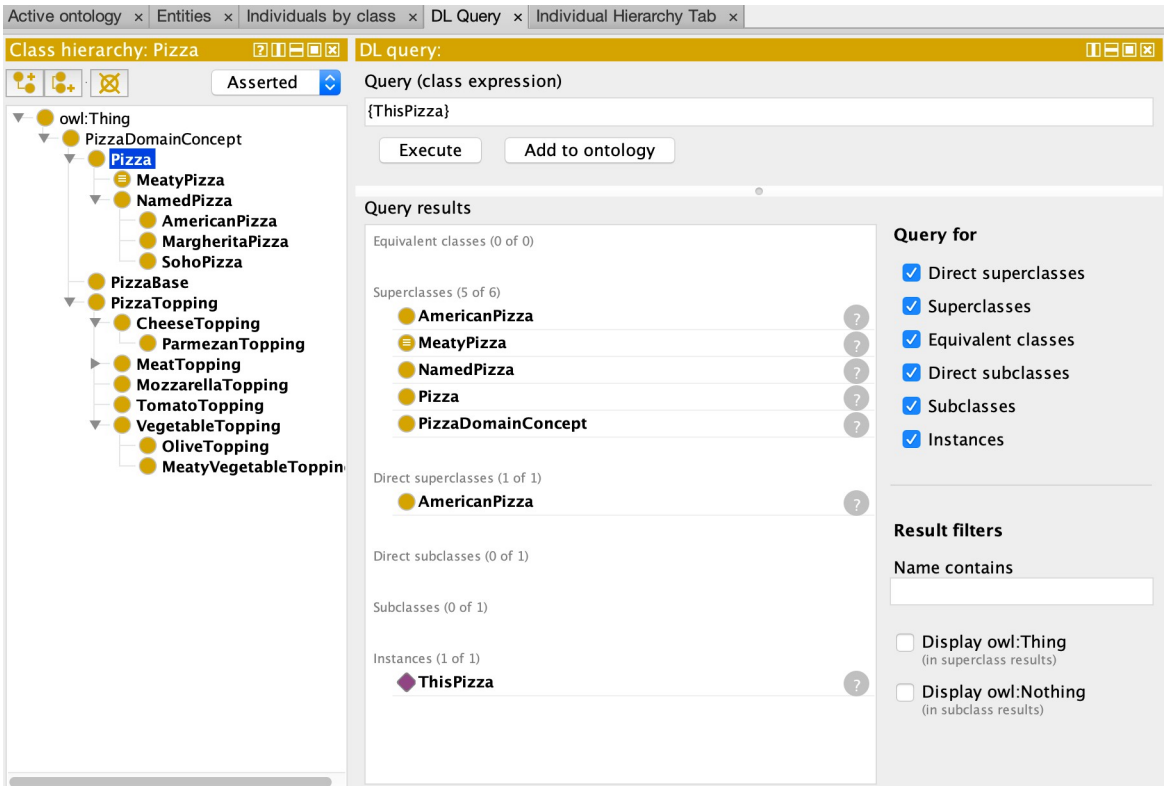
2 Create an individual in the `Individuals by class` Tab. Select Types and then choose **`AmericanPizza`**.



3 Now query the **`AmericanPizza`** in the DL query.



4 You may also query the nominal **`{ThisPizza}`**.



After you follow every step,

1. Add two more **`Pizza`**s, two more PizzaBases, two more **`PizzaTopping`**s, and add at least four more properties. (It would be fun to use ingredients from your home countries.) Explain in English what classes and properties are added.
2. Add **`Nationality`** as a concept to the ontology and introduce necessary relations. **AmericanPizza** and **`SohoPizza`** must have **`America`** as nationality. **`MargheritaPizza`** is from **`Italy`**.
3. Execute at least two (non-trivial) DL queries of your own choice. [NOTE: Take a screen shot of the results, as you will need to submit it to complete the assignment in Coursera.]
4. Save your OWL file. [NOTE: You will need it to complete the assignment in Coursera.]

If you're interested, a more complete tutorial (with an old version of Protégé) is available at

http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf