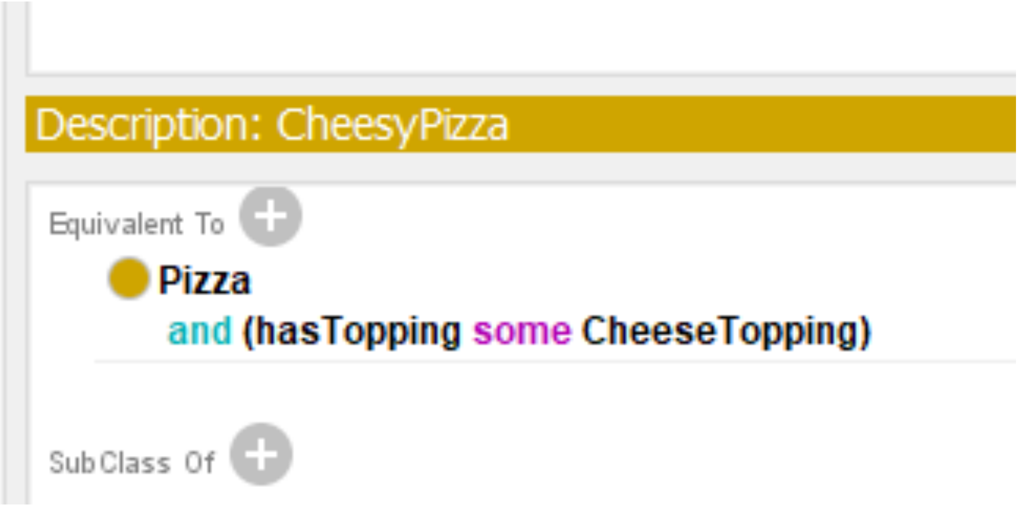
Converting conditions

Convert the necessary conditions for CheesyPizza into necessary & sufficient conditions

- 1. Ensure that CheesyPizza is selected in the class hierarchy.
- 2. In the `Edit' menu select `Convert to defined class'.



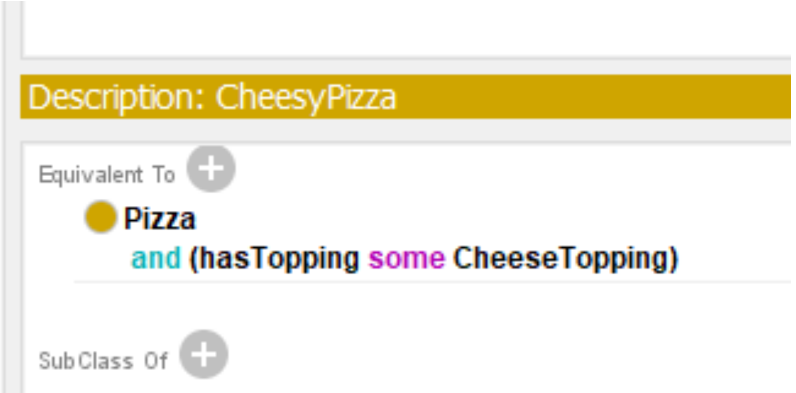
Description: CheesyPizza

Equivalent To

Pizza
and (hasTopping some CheeseTopping)

SubClass Of

Defined class



Description: CheesyPizza

Equivalent To

Pizza
and (hasTopping some CheeseTopping)

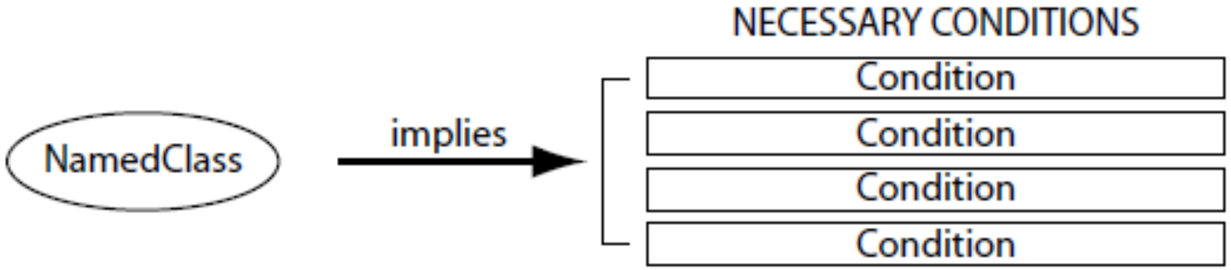
SubClass Of

- A.k.a. ‘**equivalent**’ classes in Protégé
- necessary **AND** sufficient conditions
- allows deduction in two directions

- != primitive class (necessary conditions, SubClassOf in our work)

- **Remember:**
  necessary conditions != necessary AND sufficient conditions.

Image in the Tutorial says it all



NECESSARY CONDITIONS

NamedClass

implies

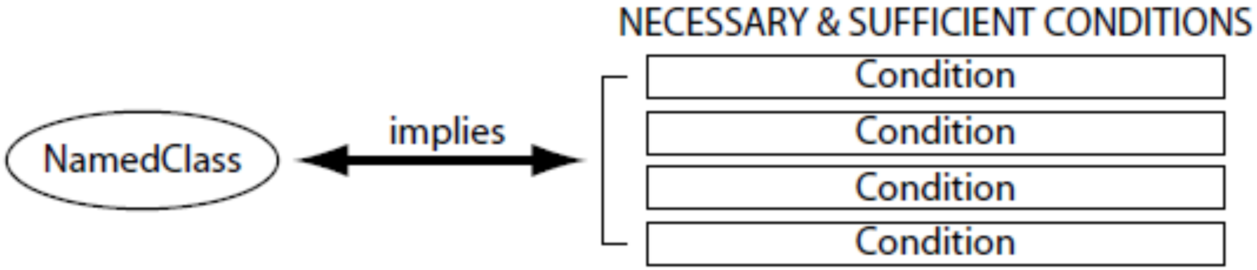
Condition

Condition

Condition

Condition

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.



NECESSARY & SUFFICIENT CONDITIONS

NamedClass

implies

Condition

Condition

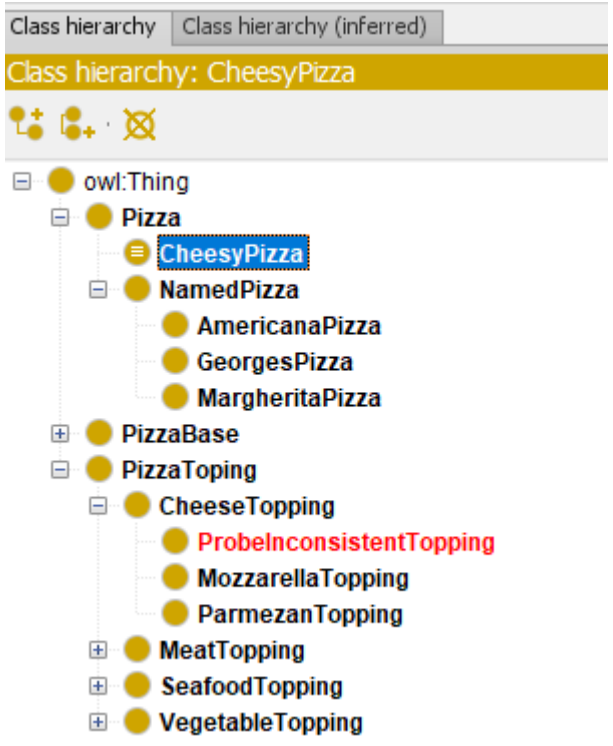
Condition

Condition

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.

Use the reasoner to automatically compute the subclasses of CheesyPizza

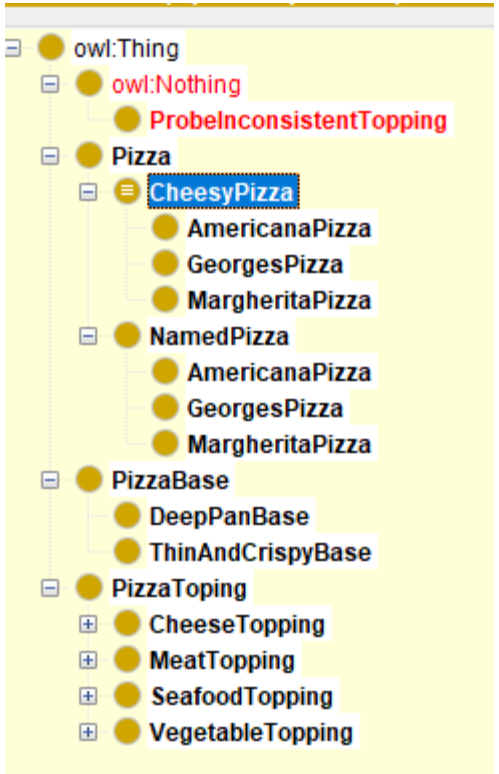
- Sync (or restart)
- Asserted vs. inferred hierarchies



Class hierarchy | Class hierarchy (inferred)

Class hierarchy: CheesyPizza

- owl:Thing
  - Pizza
    - CheesyPizza
    - NamedPizza
      - AmericanaPizza
      - GeorgesPizza
      - MargheritaPizza
  - PizzaBase
  - PizzaToping
    - CheeseTopping
      - ProbeInconsistentTopping
      - MozzarellaTopping
      - ParmezanTopping
    - MeatTopping
    - SeafoodTopping
    - VegetableTopping



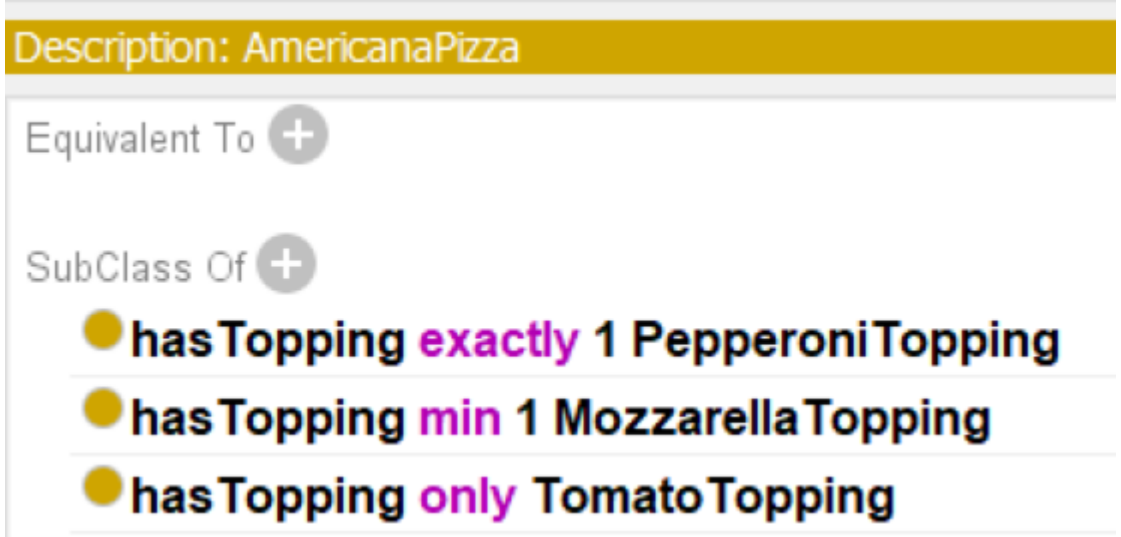
- owl:Thing
  - owl:Nothing
    - ProbeInconsistentTopping
  - Pizza
    - CheesyPizza
      - AmericanaPizza
      - GeorgesPizza
      - MargheritaPizza
    - NamedPizza
      - AmericanaPizza
      - GeorgesPizza
      - MargheritaPizza
  - PizzaBase
    - DeepPanBase
    - ThinAndCrispyBase
  - PizzaToping
    - CheeseTopping
    - MeatTopping
    - SeafoodTopping
    - VegetableTopping

Restrictions

- 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.
- 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**.
- Cardinality restrictions (car has min or exactly 4 wheels; max 8)

Trying it out (back to the main one)

- Do these ONE AT A TIME – and let me know - which make sense?



Description: AmericanaPizza

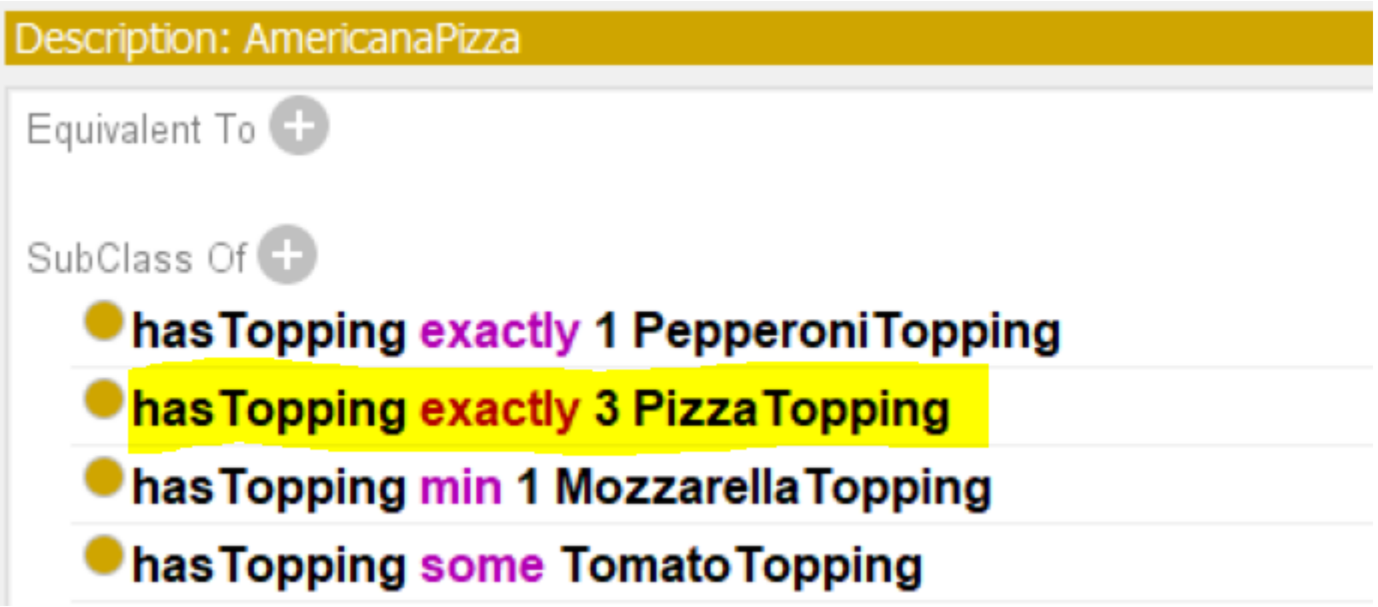
Equivalent To

SubClass Of

- hasTopping exactly 1 PepperoniTopping
- hasTopping min 1 MozzarellaTopping
- hasTopping only TomatoTopping

Be careful with cardinality

- When in doubt => some



Description: AmericanaPizza

Equivalent To

SubClass Of

- hasTopping exactly 1 PepperoniTopping
- hasTopping exactly 3 PizzaTopping
- hasTopping min 1 MozzarellaTopping
- hasTopping some TomatoTopping

Pizza **must have a** PizzaBase

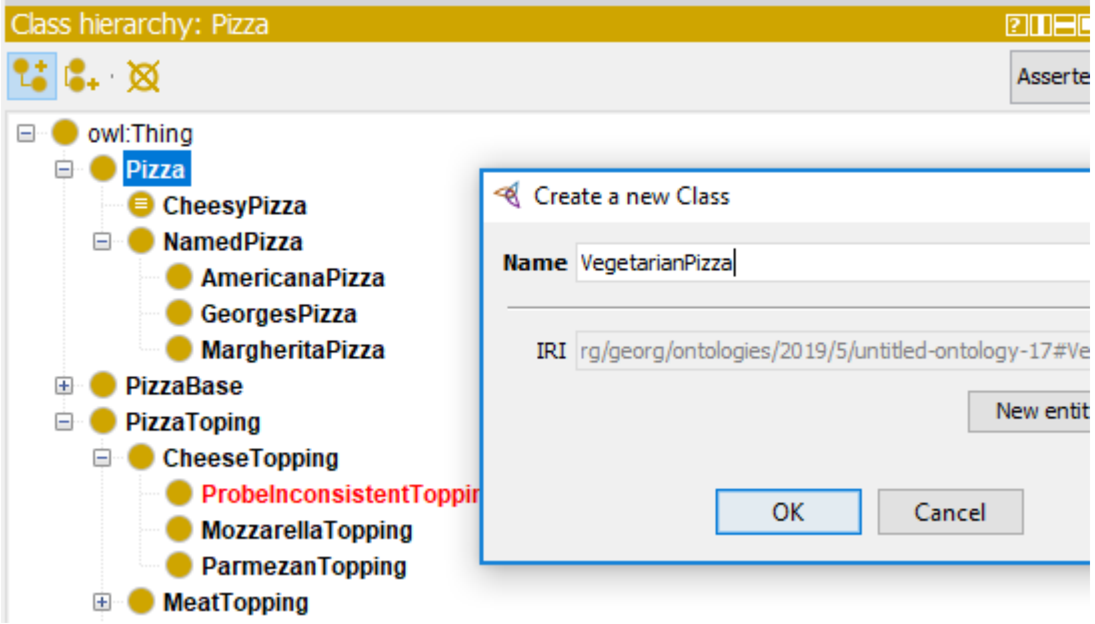
What type of condition is this in your ontology?

Where we left off

- **Create a class describing a** VegetarianPizza
- **Think about any position in hierarchy, necessary restrictions/modifications etc.**

Creating VegetarianPizza

- **Follow these steps**



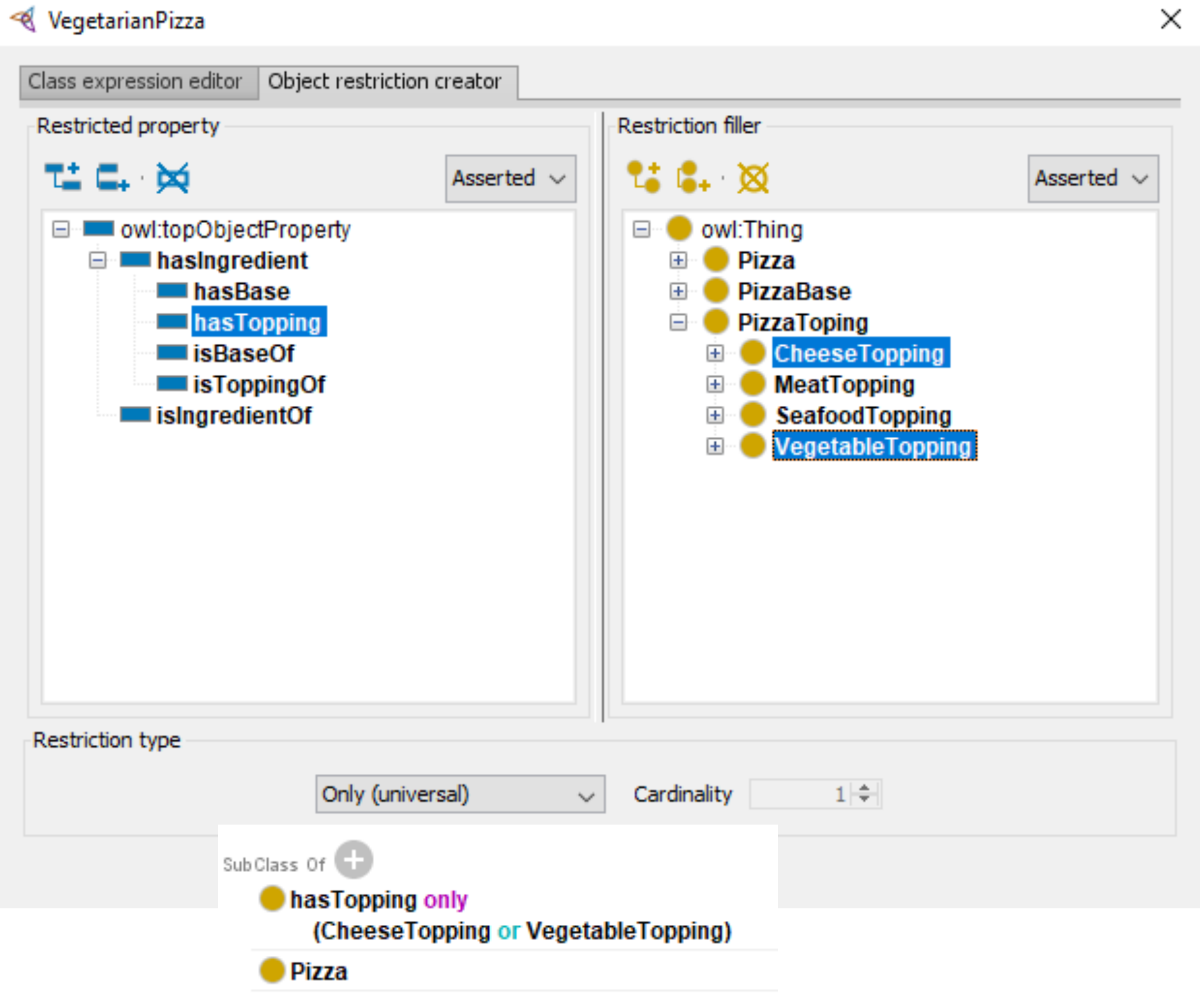
Class hierarchy: Pizza

- owl:Thing
  - Pizza
    - CheesyPizza
    - NamedPizza
      - AmericanaPizza
      - GeorgesPizza
      - MargheritaPizza
  - PizzaBase
  - PizzaToping
    - CheeseTopping
      - ProbeInconsistentToppir
      - MozzarellaTopping
      - ParmezanTopping
    - MeatTopping

Create a new Class

Name VegetarianPizza

IRI rg/georg/ontologies/2019/5/untitled-ontology-17#Ve



VegetarianPizza

Restricted property

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

Restriction filler

- owl:Thing
  - Pizza
  - PizzaBase
  - PizzaToping
    - CheeseTopping
    - MeatTopping
    - SeafoodTopping
    - VegetableTopping

Restriction type: Only (universal)

Cardinality: 1

SubClass Of

- hasTopping only (CheeseTopping or VegetableTopping)
- Pizza

- **What is different?**

Meaning

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

- “Exception” : The class VegetarianPizza also contains individuals that are Pizzas and do not participate in **any** hasTopping relationships.

Convert class

- RATIONALE ALERT!!!!!!!!!!!!

- Our VegetarianPizza definition is pretty solid. Therefore, we can:

- **Convert the necessary conditions for** VegetarianPizza **into necessary & sufficient conditions**

1. Ensure that VegetarianPizza is selected in the class hierarchy.
2. In the `Edit' menu select `Convert to defined class’.

- Or right click, `Convert to defined class’.
- or CTRL-D

Try the reasoner

- We have the definition of VegetarianPizza

- We have at least one pizza that meets the requirements

- **Did it work? Why (not)? Should it (have) work(ed)?**
- **Submit your thoughts to Week 7 formative 2**

Open and Closed World Assumption

- What did we say open world is?
- The open world assumption means that we cannot assume something doesn't exist until it is explicitly stated that it does not exist

- We believe that MargheritaPizza should be a subclass VegetarianPizza

Rationale

- 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 as well.
- 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** on the hasTopping property.

Add a closure axiom – the **hard** way

- **Add a closure axiom on the** hasTopping **property for** MargheritaPizza

1. Make sure that MargheritaPizza is selected in the class hierarchy on the `Classes’ tab.

2. Press the `Add' icon next to the `SubClass of' section of the `Class Description' view to open the edit text box (Class expression editor).

3. **Type** hasTopping as the property to be restricted.

4. **Type** `only' to create the universal restriction.

5. **Open brackets and type MozzarellaTopping or TomatoTopping close bracket.**

6. **Press** `OK' to create the restriction and add it to the class MargheritaPizza.

Meaning of the closure axiom

- **In natural language**
- **Literally just what we did**
- **Someone?**

Meaning of the closure axiom

What we had

- **If** an individual is a member of the class MargeritaPizza
- **Then** it must be a member of the class Pizza,
  and it must have at least one topping that
    is a kind of MozzarellaTopping
  and it must have at least one topping that
    is a member of the class TomatoTopping

- **Is this enough??**

Meaning of the closure axiom

What we have

- **If** an individual is a member of the class MargeritaPizza
- **Then** it must be a member of the class Pizza,
    - **and** it must have at least one topping that
        - **is a** kind of MozzarellaTopping
    - **and** it must have at least one topping that
        - **is a** member of the class TomatoTopping
    - **and** the toppings must only be
        - **kinds of (**MozzarellaTopping **or** TomatoTopping**)**

Closure axiom Definition

- For example, the closure axiom on the ∀ hasTopping property for MargheritaPizza is a universal restriction that acts along the hasTopping property, with a filler that is the **union** of MozzarellaTopping and also TomatoTopping.

- Simply:
- MozzarellaTopping **U** TomatoTopping

@Class

- **Why** do you think we have :

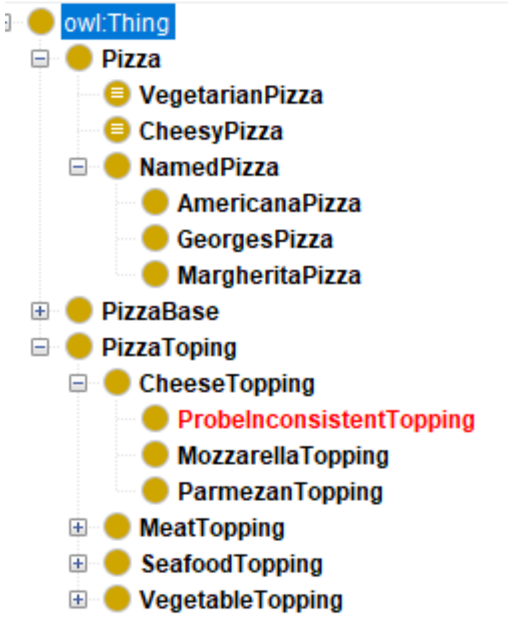
  hasTopping only (MozzarellaTopping or TomatoTopping)
  hasTopping some MozzarellaTopping
  hasTopping some TomatoTopping

- And not:

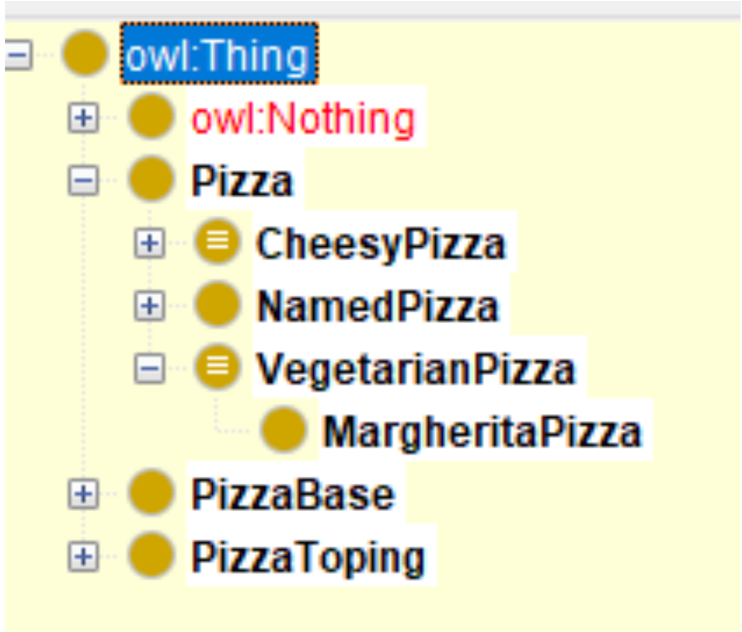
  hasTopping only (MozzarellaTopping or TomatoTopping)

Test it!

- **Sync Reasoner**



- owl:Thing
  - Pizza
    - VegetarianPizza
    - CheesyPizza
    - NamedPizza
      - AmericanaPizza
      - GeorgesPizza
      - MargheritaPizza
  - PizzaBase
  - PizzaToping
    - CheeseTopping
      - ProbeInconsistentTopping
      - MozzarellaTopping
      - ParmezanTopping
    - MeatTopping
    - SeafoodTopping
    - VegetableTopping



- owl:Thing
  - owl:Nothing
  - Pizza
    - CheesyPizza
    - NamedPizza
    - VegetarianPizza
      - MargheritaPizza
  - PizzaBase
  - PizzaToping

- **BOOOOM!!!!** **(hopefully)**

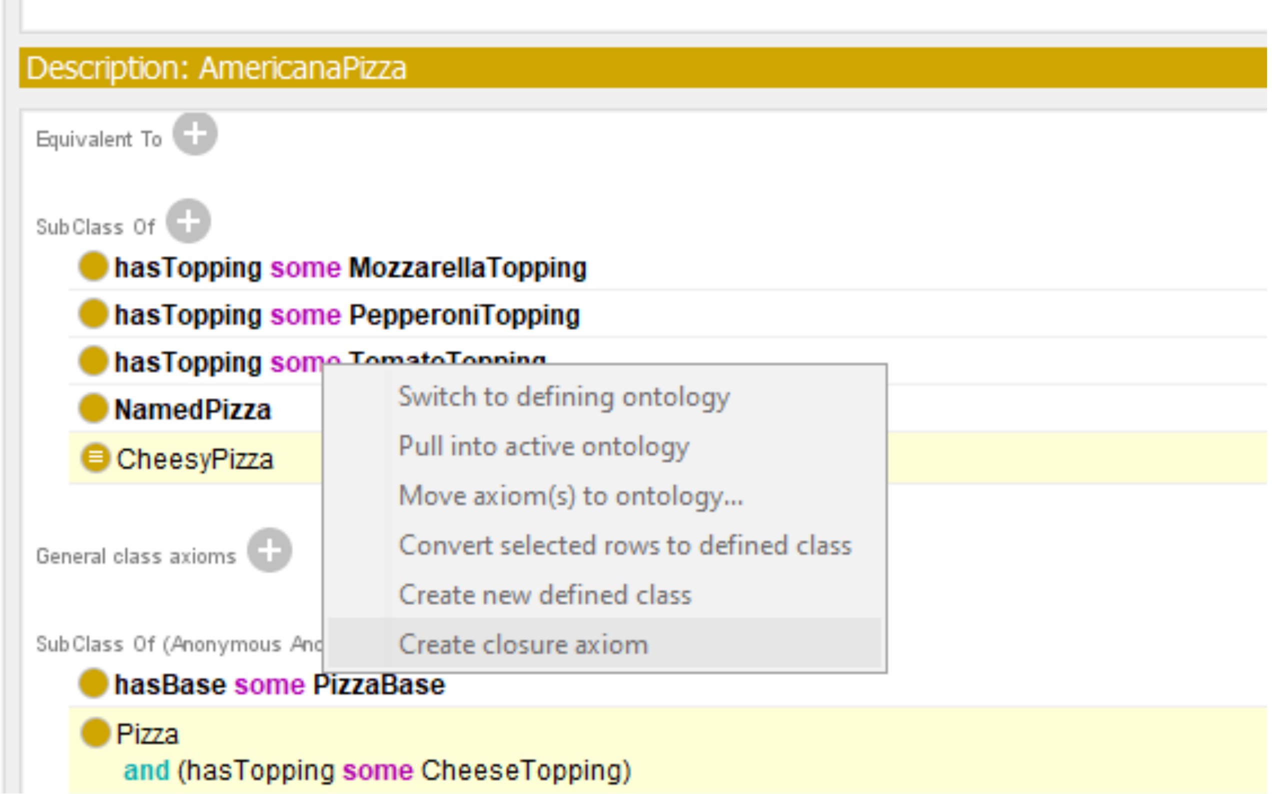
Exercise

- Add closure axiom for **AmericanaPizza**

Auto closureAxiom (the easy way)

- For AmericanaPizza
- Left click to select **(in the white)**
- Right click for closure (ALL)

- **Sync**



Description: AmericanaPizza

Equivalent To

SubClass Of

hasTopping some MozzarellaTopping

hasTopping some PepperoniTopping

hasTopping some TomatoTopping

NamedPizza

CheesyPizza

General class axioms

SubClass Of (Anonymous Anc

hasBase some PizzaBase

Pizza
and (hasTopping some CheeseTopping)

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

Value Partitions

- Used to refine our descriptions of various classes.
- Value Partitions are ‘not part of OWL’, they are a “design pattern”.
- Design patterns in ontology design are analogous to design patterns in object oriented programming
  (= solutions to modelling problems that have occurred over and over)
- These design patterns have been developed by experts and are now recognised as proven solutions for solving common modelling problems.
- **closed set of values for a property**

If we wanted to make a Value Partition: SpicinessValuePartition

This is the approach - Not an exercise – do not do this

- 1. Create a class to represent the **ValuePartition**. For example to represent a `spiciness' ValuePartition we might create the class **SpicinessValuePartition**.
- 2. Create subclasses of the ValuePartition to represent the possible options for the ValuePartition. For example we might create the classes **Mild**, **Medium** and **Hot** as subclasses of the SpicynessValuePartition class.
- 3. Make the subclasses of the ValuePartition class **disjoint (why?)**.

Provide a **covering** axiom to make the list of value types **exhaustive**

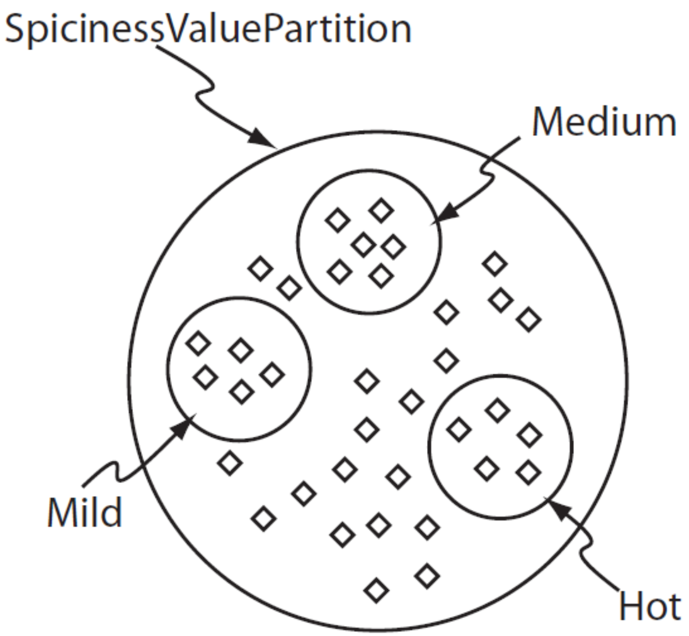
This is the approach - Not an exercise – do not do this

- 4. Create an object property for the ValuePartition. For example, for our spiciness ValuePartition, we might create the property **hasSpiciness**.
- 5. Make the property **functional (Class: why?)**.
- 6. Set the **range** of the property as **the ValuePartition** class. For example for the **hasSpiciness** property the range would be set to **SpicinessValuePartition**.

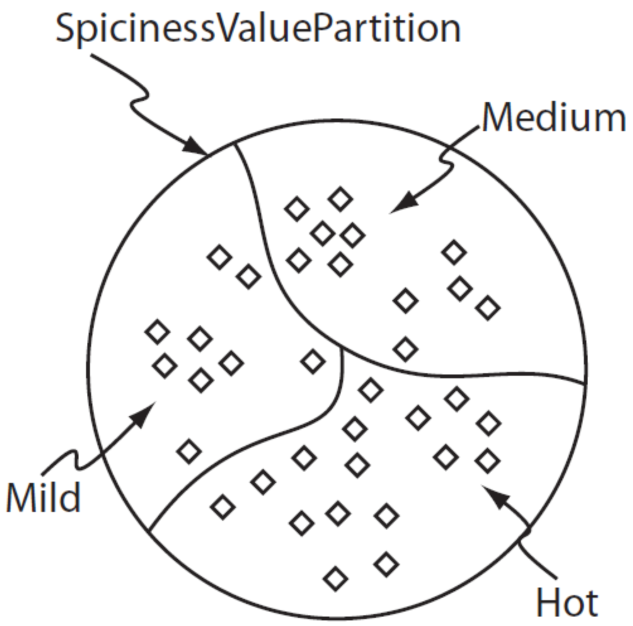
Value Partitions

- What they may look like schematically

- Any comment on the class – subclass difference?



Without a covering axiom

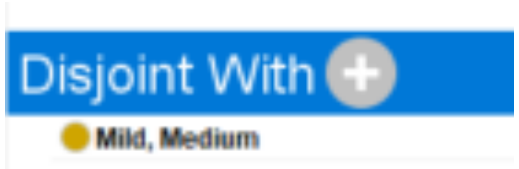


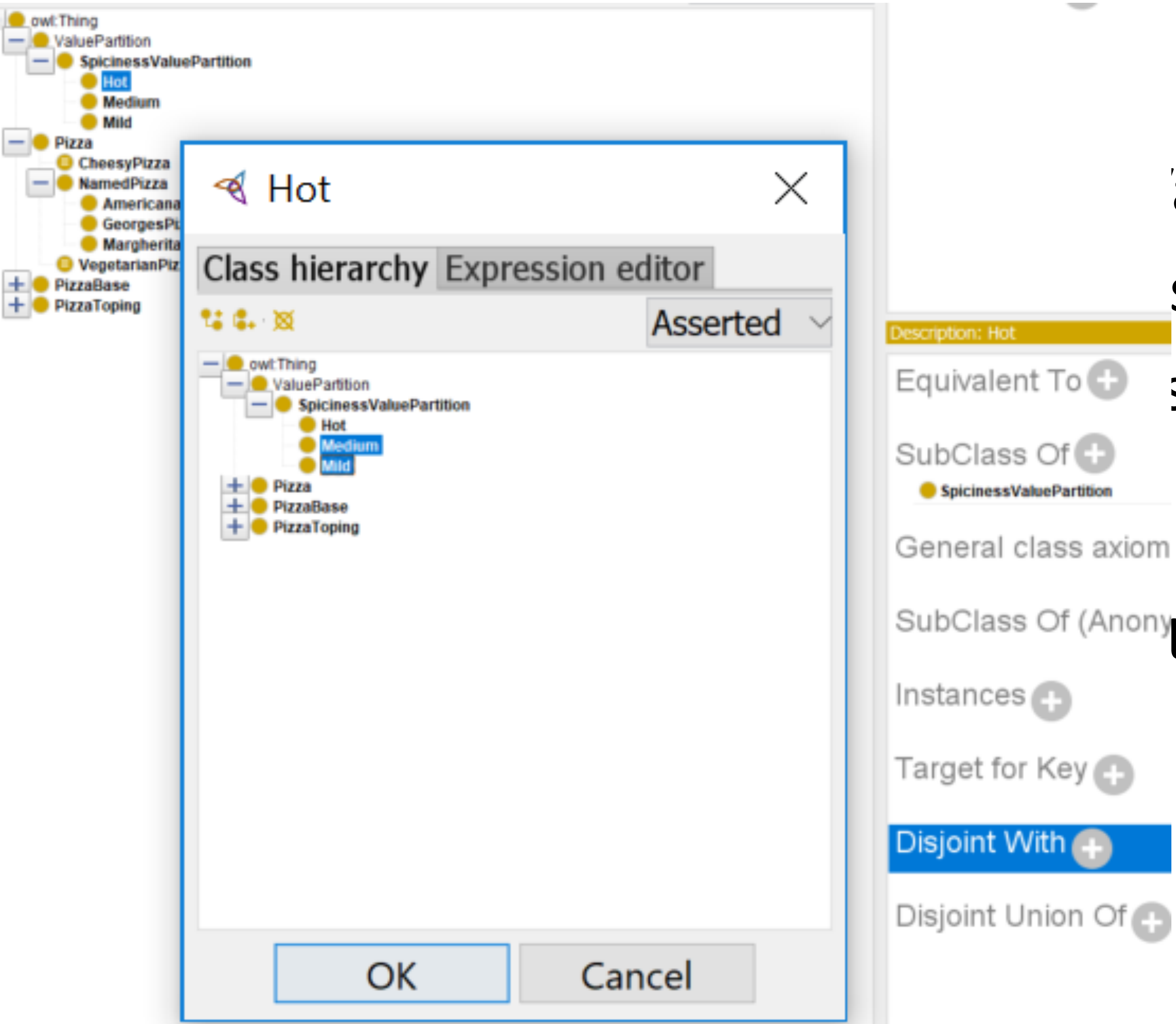
With a covering axiom (SpicinessValuePartition is covered by Mild, Medium, Hot)

Value Partitions - Exercise

- 1. Create a new class as a sub class of Thing called ValuePartition.
- 2. Create a sub class of ValuePartition called SpicinessValuePartition.
- 3. Create three new classes as subclasses of SpicinessValuePartition. Name these classes Hot, Medium, and Mild.
- 4. Make the classes Hot, Medium, and Mild disjoint from each other. You can do this by selecting the class Hot, and selecting `Make all primitive siblings disjoint' from the `Edit' menu.
  (**or you can try the traditional way**)

Value Partitions

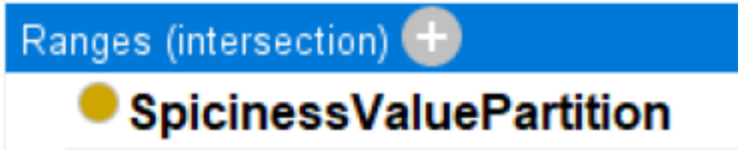




- 1. Create : aluePartition.
- 2. Create : ssValuePartition.
- 3. Create ssValuePartition. Name the
- 4. Make tl from each other. You can d ting `Make all primitive : (or you ca

Value Partitions

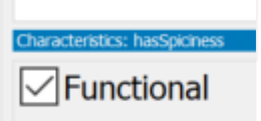
- 5. In the `Object Property Tab' create a new Object Property called **hasSpiciness**. Set the **range** of this property to **SpicinessValuePartition**.



Ranges (intersection)

● SpicinessValuePartition

- 6. Make this new property functional



Characteristics: hasSpiciness

☑ Functional

- 7. Add a covering axiom to the SpicinessValuePartition. Highlight SpicinessValuePartition in the class hierarchy, in the “Equivalent To” section of the class description view select the “Add” icon and **type** *Hot or Medium or Mild* in the dialog box.

Adding Spiciness to Pizza Toppings

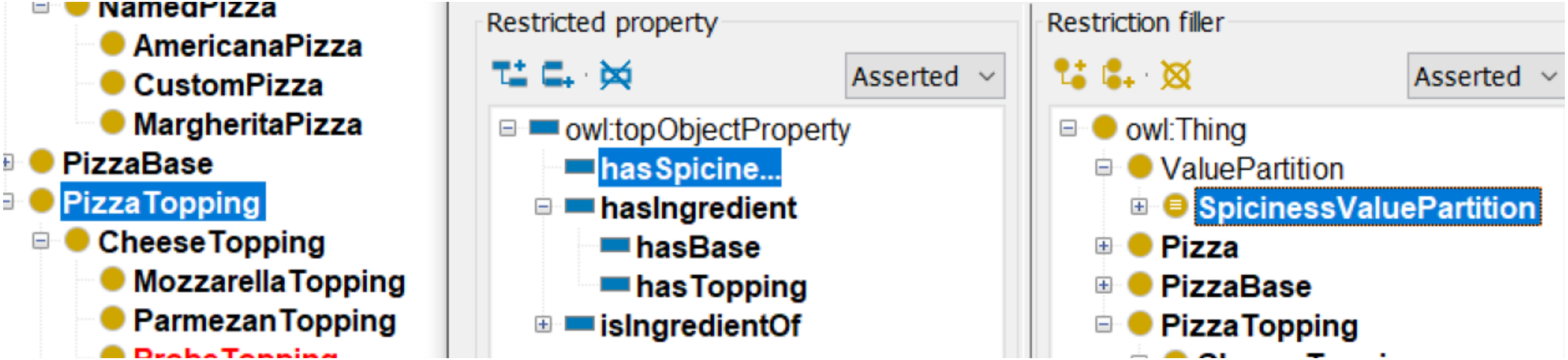
- We can now use the SpicinessValuePartition to describe the spiciness of our pizza toppings. To do this we will add an **existential restriction** to each kind of PizzaTopping to state it's spiciness.
- Restrictions will take the form:

  hasSpiciness ***some*** **SpicynessValuePartition**

  where **SpicinessValuePartition** will be one of Mild, Medium, or Hot

Adding Spiciness to Pizza Toppings

- Can also do the general case first (why? Is it correct?)



- NamedPizza
  - AmericanaPizza
  - CustomPizza
  - MargheritaPizza
- PizzaBase
- PizzaTopping
  - CheeseTopping
    - MozzarellaTopping
    - ParmezanTopping
    - ProbeTopping

Restricted property

Asserted

- owl:topObjectProperty
  - hasSpicine...
  - hasIngredient
    - hasBase
    - hasTopping
  - isIngredientOf

Restriction filler

Asserted

- owl:Thing
  - ValuePartition
    - SpicinessValuePartition
  - Pizza
  - PizzaBase
  - PizzaTopping

hasSpiciness restrictions on PizzaToppings

- 1. Make sure that “MozarellaTopping” is selected in the class hierarchy.
- 2. Use the “Add” icon on the “Subclass Of”
- 3. Select the “Class expression editor” tab.
- 4. Type (or select) **hasSpiciness some Mild** to create the existential restriction.
- 5. Press “OK” to create the restriction -- if there are any errors, the restriction will not be created, and the error will be highlighted in red.
- 6. **Repeat this for each of the bottom level PizzaToppings (those that have no subclasses).**