Domain/Range Flexibility

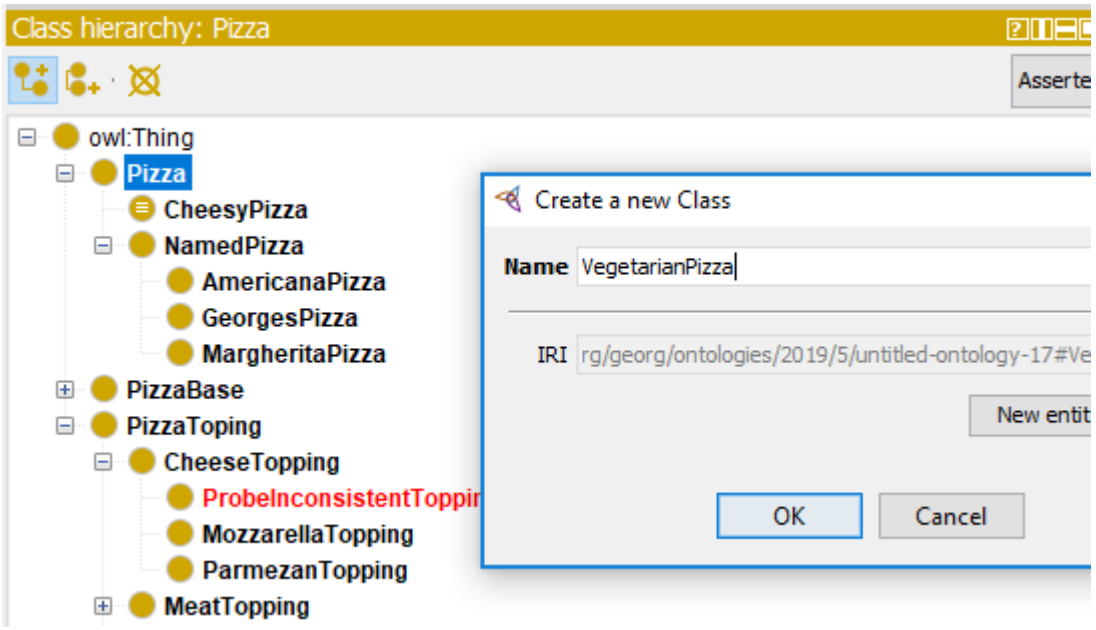
- It is possible to **specify multiple classes** as the range for a property.
- If multiple classes are specified in Protege the range of the property is interpreted to be the **intersection** of the classes.
- Sometimes it makes sense. For example, if the range of a property has the classes Man and Woman listed in the range view, the range of the property will be interpreted as Man intersection Woman
- **However, you can specify logical and/or to satisfy both union and intersection**

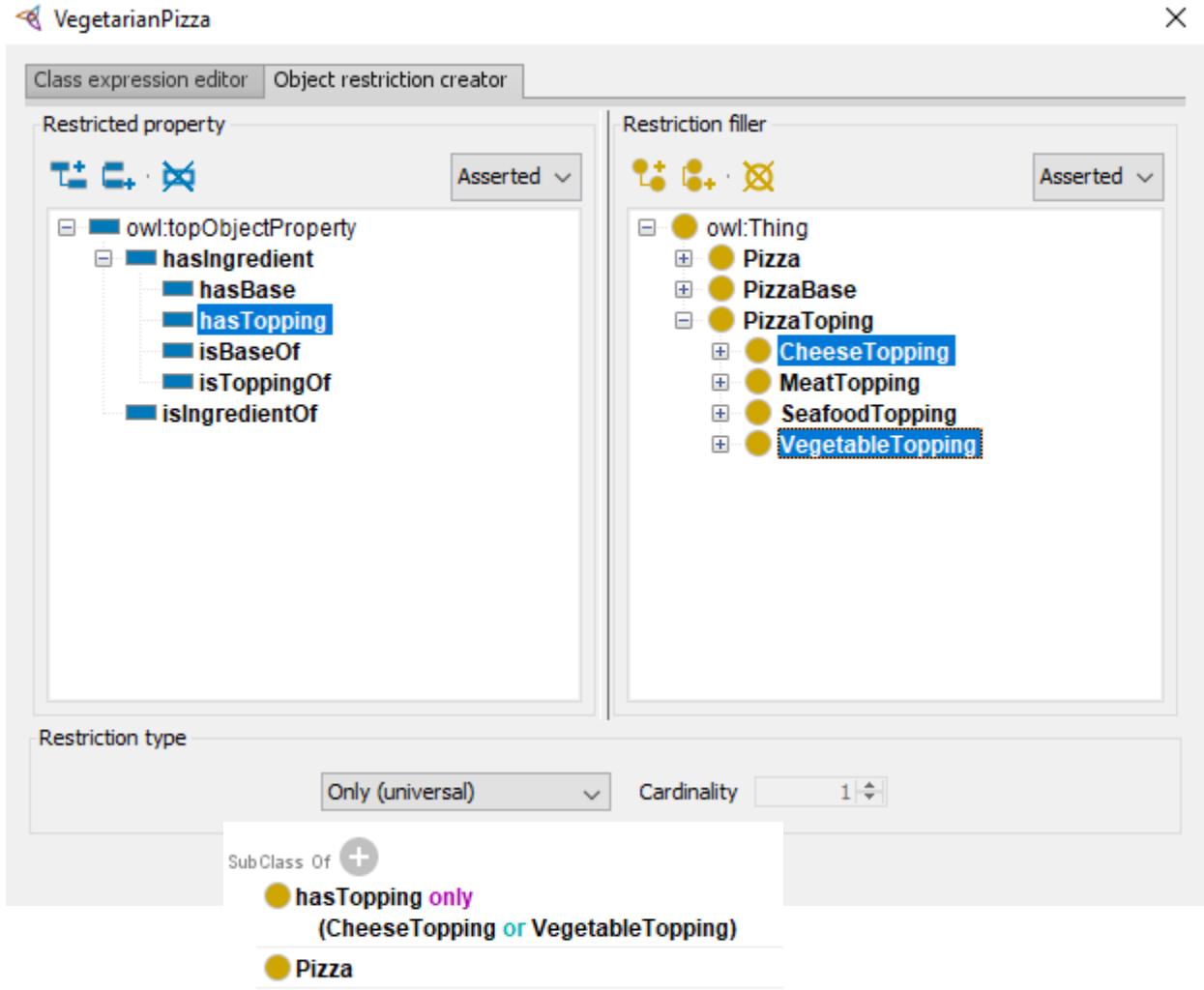
Domain/Range Flexibility

- Download the ‘pick up’ file for 27th June

- Can you observe anything different to last week?

Recap: Creating VegetarianPizza





- **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.

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.

Closure axiom Definition

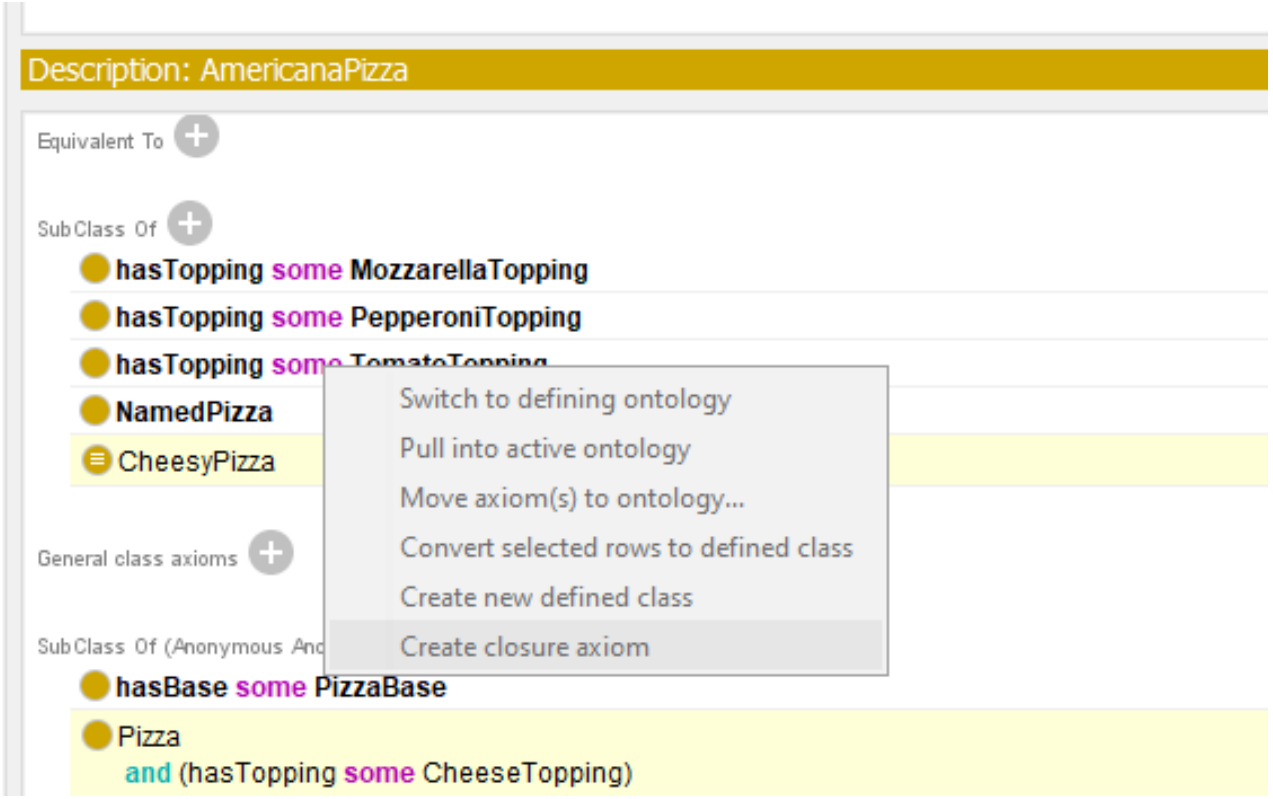
- For example, the closure axiom on the ∀ hasTopping property for MargheritaPizza is a universal restriction that acts along the hasTopping property, with a filler that is the **union** of MozzarellaTopping and also TomatoTopping.

- Simply:
- MozzarellaTopping ∪ TomatoTopping

Auto closureAxiom (the easy way)

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

- **Sync**



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.

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?)**.

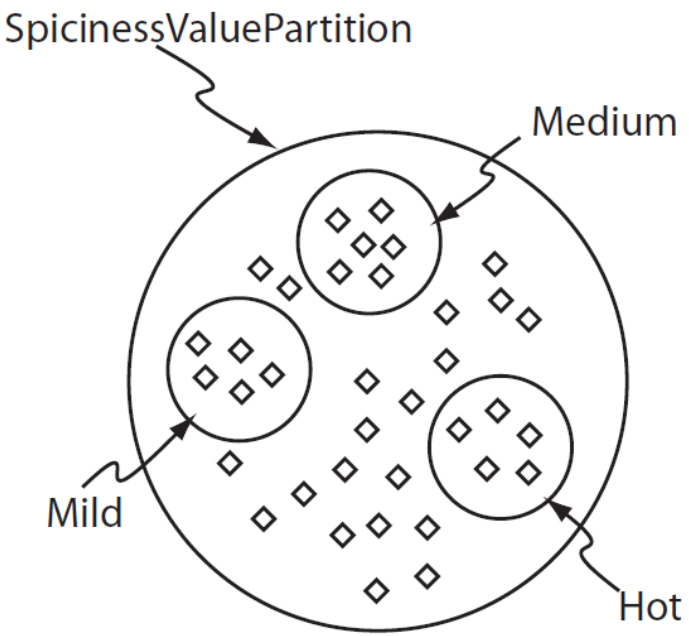
Provide a **covering** axiom to make the list of value types **exhaustive**

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

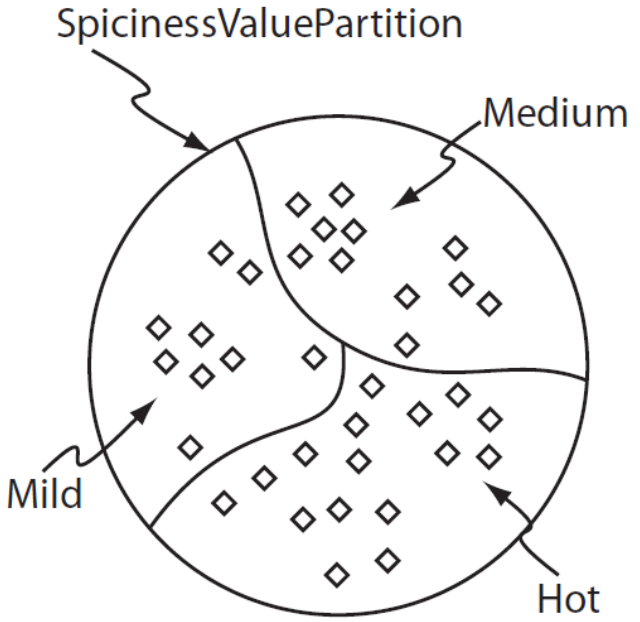
- 4. Create an object property for the ValuePartition. For example, for our spiciness ValuePartition, we might create the property **hasSpiciness**.
- 5. Make the property **functional (Class: why?)**.
- 6. Set the **range** of the property as **the ValuePartition** class. For example for the **hasSpiciness**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

hasTopping some (PizzaTopping and (hasSpiciness some Hot))

- Our description of a SpicyPizza above says that:

  all members of SpicyPizza
  - are Pizzas
  and
  -have at least one topping
  that has a Spiciness of Hot.

It also says that anything that is a Pizza and has at least one topping that has a spiciness of Hot **is a SpicyPizza**

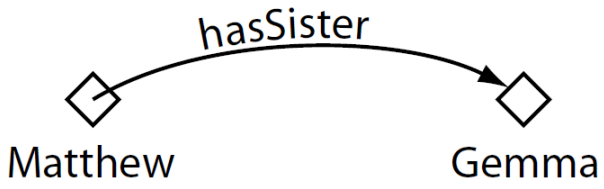
=> @Class: what option led to this?

Datatype Properties

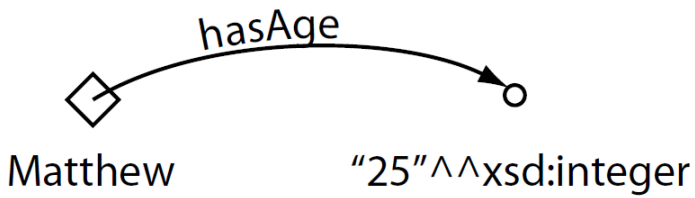
- != object properties = relationships between individuals

- Datatype properties link an individual to an XML Schema Datatype value or an RDF literal.
- In other words, they describe relationships **between an individual and data values**.

Datatype Properties



An object property linking the individual Matthew to the individual Gemma



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

Create a datatype property called **hasCalorificContentValue**

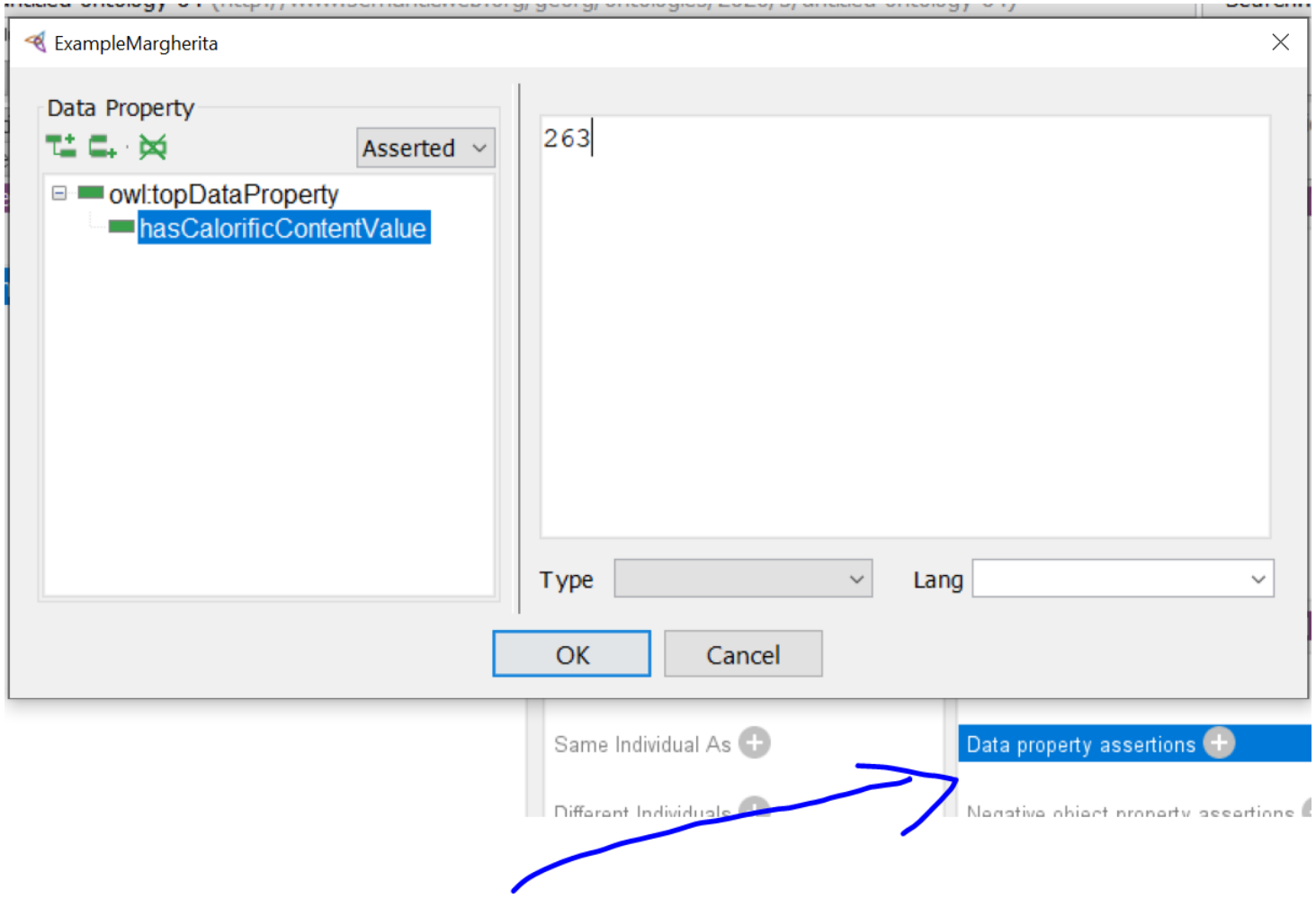
- Entity => **Data Properties** (or DataType Properties; version-dependant)
- **Green colour**
- Use the `Add Sub Property' button to create a new Datatype property called hasCalorificContentValue

Creating Individuals

1. Entities => Individuals (purple colour)
2. Press the `Add individual' button and create an individual called **Example-Margherita**
3. In the `Individual Description view' add a **type** of MargheritaPizza (2 ways)
4. In the `Property assertions view' add a `Data property assertion' and in the dialog ensure hasCalorificContentValue is selected as the property, integer is selected as the type and a value of **263** is entered.
5. **Create several more example pizza individuals with different calorie contents including an instance of QuattroFormaggio with 723 calories.**

Screenshot

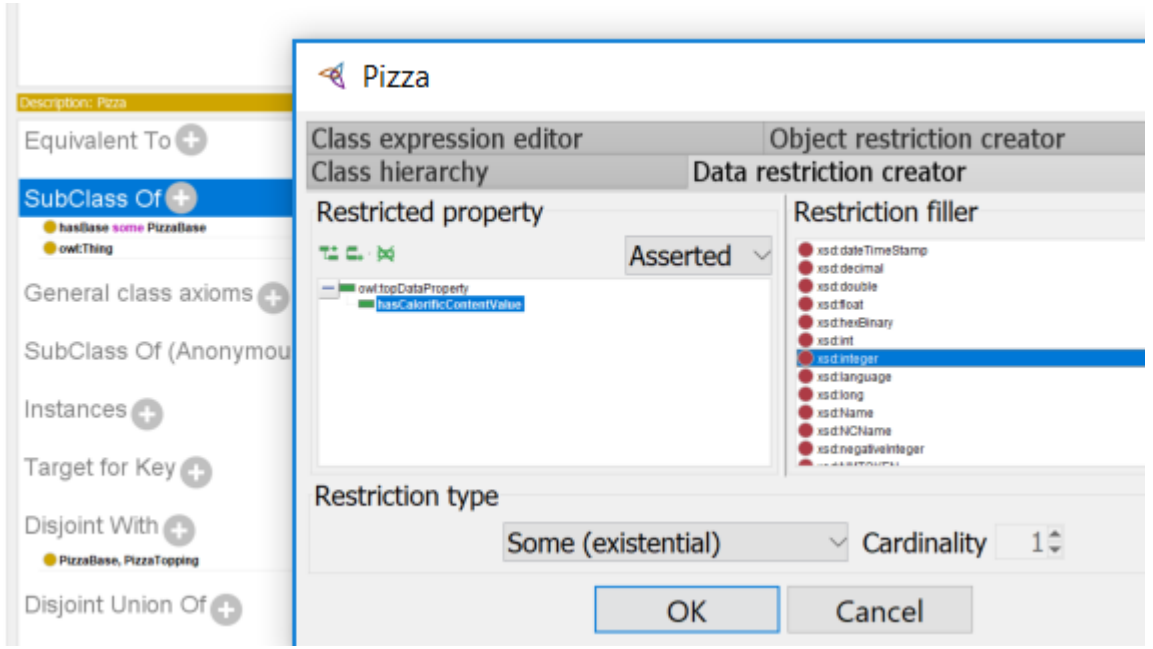
- !


ExampleMargherita
Data Property
Asserted
owl:topDataProperty
hasCalorificContentValue
263
Type
Lang
OK
Cancel
Same Individual As
Data property assertions
Different Individuals
Negative object property assertions

All Pizzas Must have a Calorific Value

- Sounds like a restriction - **a datatype restriction**
- Entities=>Classes (or Class Hierarchy)=>**Pizza**=>SubClass Of
- **Data restriction creator** tab!

- hasCalorificContentValue
  some
  xsd:integer

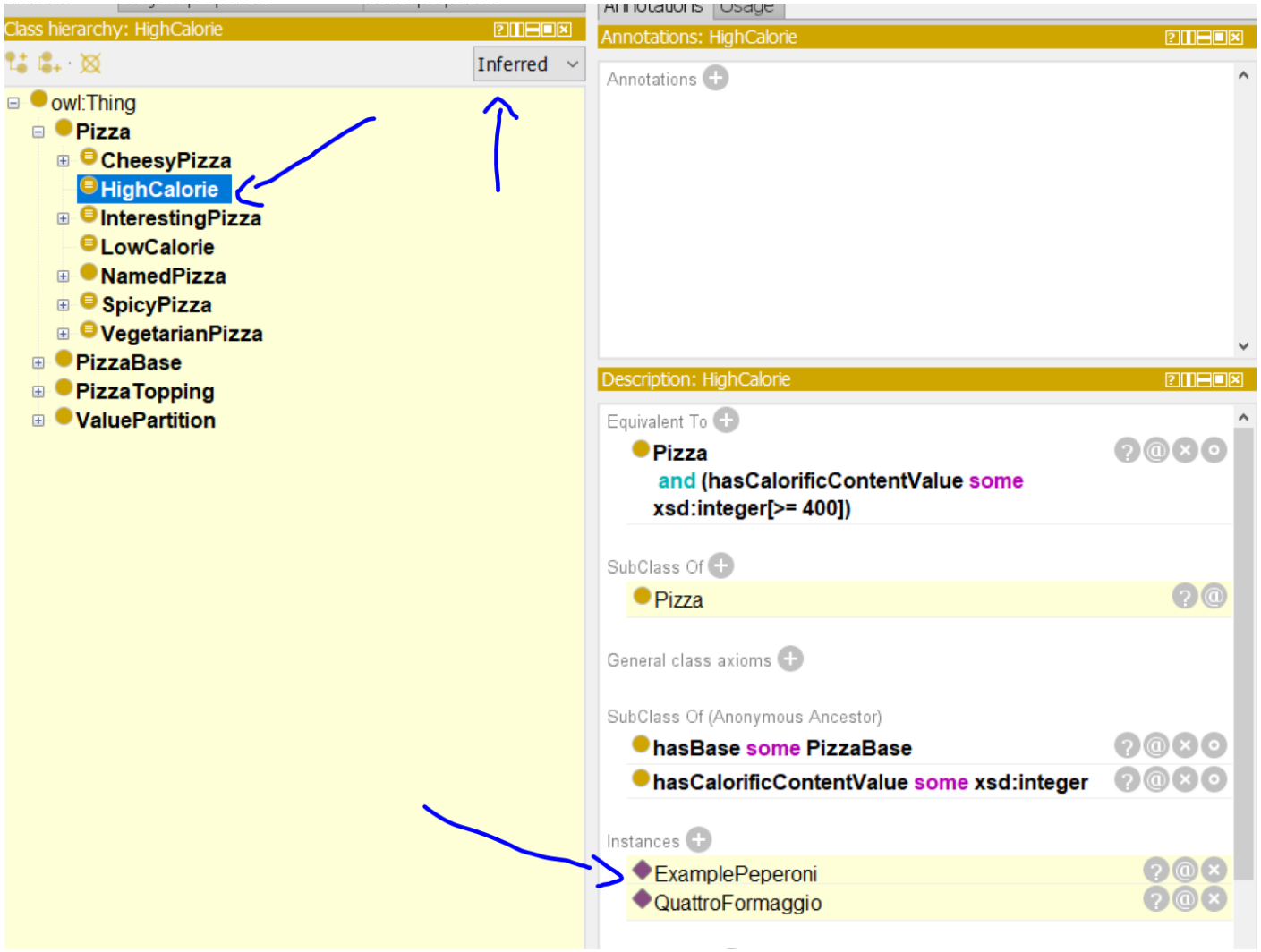


Exercise

- Create HighCaloriePizza as subclass of Pizza
- Select HighCaloriePizza
- Class description view => Subclass Of
- Class expression editor
- hasCalorificContentValue some xsd:integer[>= 400]
- Make **defined class** (menu, right click or.. CTRL-D)

Check with the reasoner

- Graph looks the same
- Check the Individuals or Instances (a.k.a. members)



Preparation & Revision

- Does anyone qualify for extra time?
- **Email me if so**

Preparation & Revision

- Series 1: True/False, Short answers (1-2 sentences)
- Series 2: Slightly longer answers (3-6 sentences)
- 3 and 4: Solve an issue **and document** what you did

- Apart from the correct answer, I can about the **why**
  => justify your answer

- **SQL is not going to be assessed; Sparql will be assessed**

Ambiguities
(‘longer answer’ example)

- Submit all ambiguities you can think of (in natural language) for the ‘Heisenberg can wait’ picture
- Let’s read into this question:
  - ‘all ambiguities’ means >=2
  - in natural language = no formal representation
  - why is something ambiguous
  (this will be phrased accordingly in the exam)

Ambiguities

- **Werner Heisenberg - Well known German theoretical physicist. Results could be about him.**
  (0.5 ambiguity since no explanation of **what else it** could be, natural language ok, no further explanation)
  => Indicative grade: F
- **German Physician Heisenberg NFL games VS Heisenberg nickname from Breaking Bad as they are both Tv shows.**
  (1 ambiguity, natural language ok, explanation ok)
  => Indicative grade: C

Ambiguities

- Heisenberg could refer to either the German physicist or the main character of the show Breaking Bad. Yet, us humans can understand that the poster refers to the main character of the show, because the sign is trying to promote the Sunday Night Football show, which probably happens to be airing at the same time as Breaking Bad.
  (1 ambiguity, natural language ok, went beyond the expected justification)
  => Indicative grade: B

Ambiguities

- 1.Heisenberg : famous physicist (principle of uncertainty) AND star of the tv show Breaking Bad **AND everybody else called Heiseinberg** 2.#snf :can be the abbreviation of many things, such as […] 3.football: the game and the actual ball […] 4. **can**: the object AND the verb […]
  (multiple ambiguities; one of which **went beyond** expectancies; one additional not discussed in class, natural language ok; is a bit formal, justification ok; did not elaborate)
  => Indicative grade: A

Mock Exam

- Blackboard => Exams => Mock exam
- We will go through the answers together at the end

What are facets? Provide an example.

- Facets are restrictions we create using properties. An example is that a Margherita Pizza must have at least one Mozzarella Topping (assuming a ‘hasTopping’ property), which is an existential restriction.
- C
- B
- A-
- A
- bonus

What are Qnames? Provide an example.

- Qualified name. Contains a prefix that has been assigned to a namespace URI.
- Example: “foo:bar” is a shorthand for the URI http://example.com/somewhere/bar

Provide two reasons pro and one con for using the TriX notation

- + Easy to read – triples are clearly annotated
- + Can name our graph
- - Takes up a lot of resources due to repetition

Which category of markup languages does HTML belong to and why?

- Descriptive markup as it labels parts of the document to provide specific instructions as to how they should be processed.

What is the difference between a defined and a primitive class? Provide an example for each.

- A defined class is a class which has both necessary and sufficient conditions. It allows deduction in two directions. It is important to note that a primitive class has only necessary conditions.
  An example of a defined class could be a parent with the N&S condition to have at least one child. A primitive class example would be a cheesy pizza with the N conditions to have some cheese topping

Provide the SPARQL query for retrieving the coach names for each of the teams that Giannis Antetokounmpo has played for. Ideally the results will be in alphabetical order and limited to the first two entries in the database. You are likely to need http://dbpedia.org/sparql and Giannis Antetokounmpo's dbpedia page

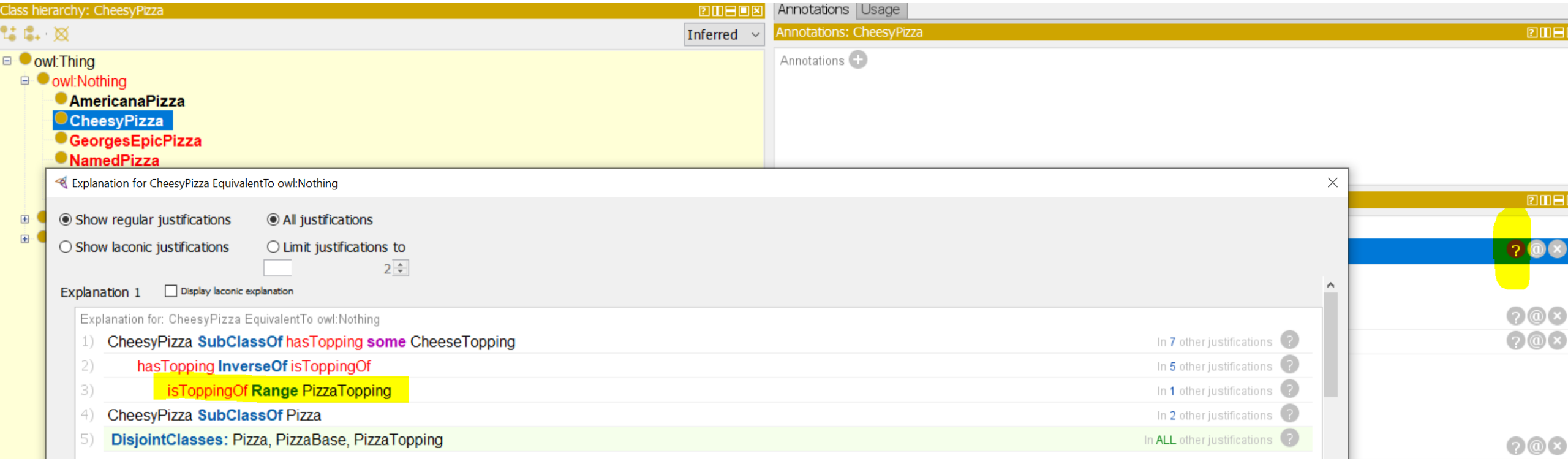
```
SELECT ?name
WHERE {
   dbr:Giannis_Antetokounmpo dbo:team ?team.
   ?team dbo:coach ?coach .
   ?coach dbp:name ?name .
}
ORDER BY ASC(?name)
LIMIT 2
```

Protégé problem

- Download the file and proceed with
- 1. Identify the logical mistakes and
  2. Edit the probe
  in order to stop the reasoner from flagging issues
- 3. Add MargheritaPizza with at least one Mozzarella
  and at least one Tomato topping
  4. Make the appropriate modifications to allow new inferences
  for CheesyPizza
- Explain what you did in comments

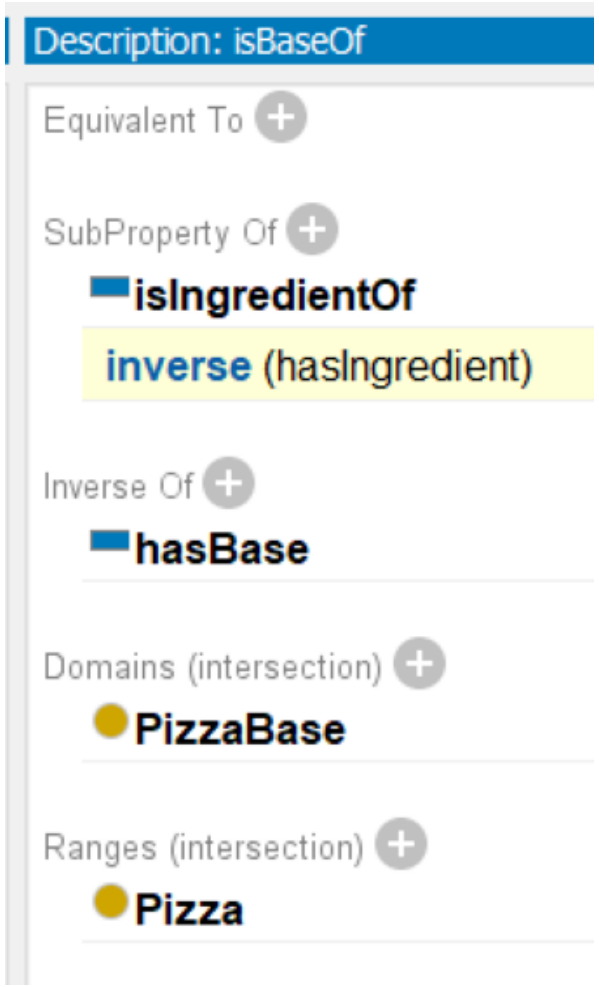
Mock Exam - Problem

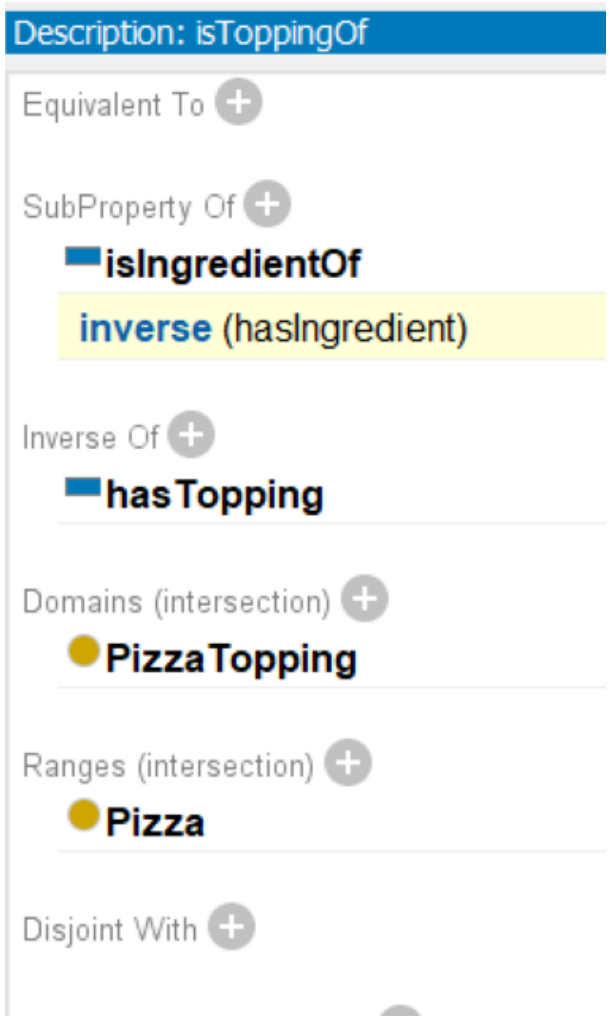
- Follow up and find something that does not make sense



1. logical issue

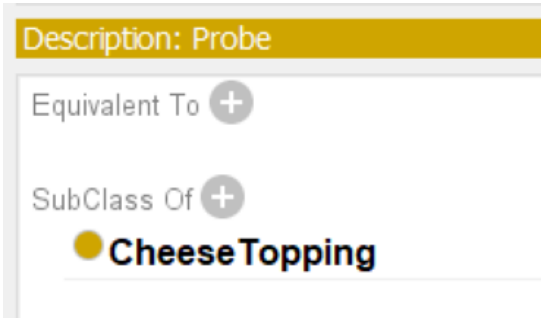
- Range/domain

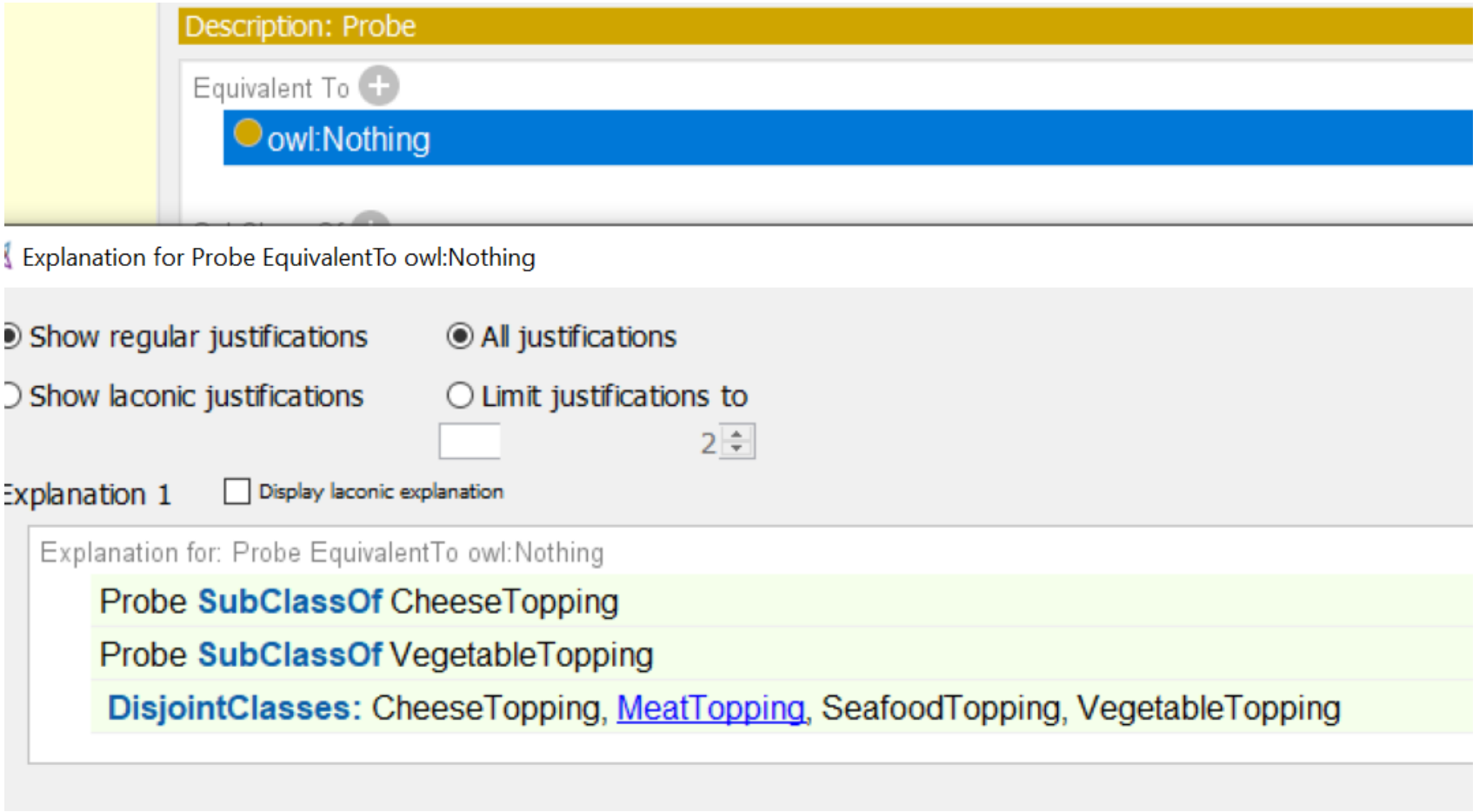




2. Probe

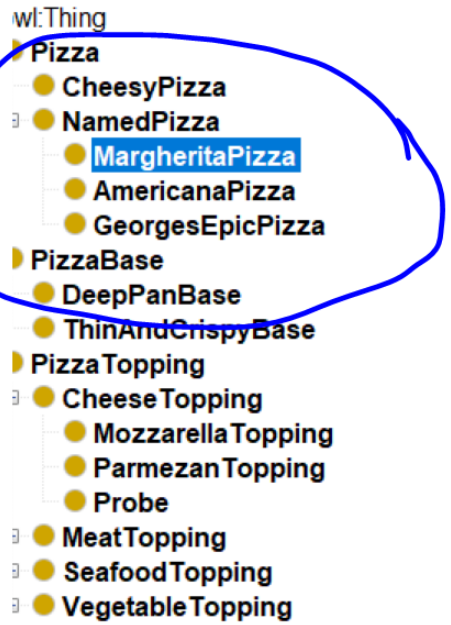
- Which of the 2 solutions makes more sense?

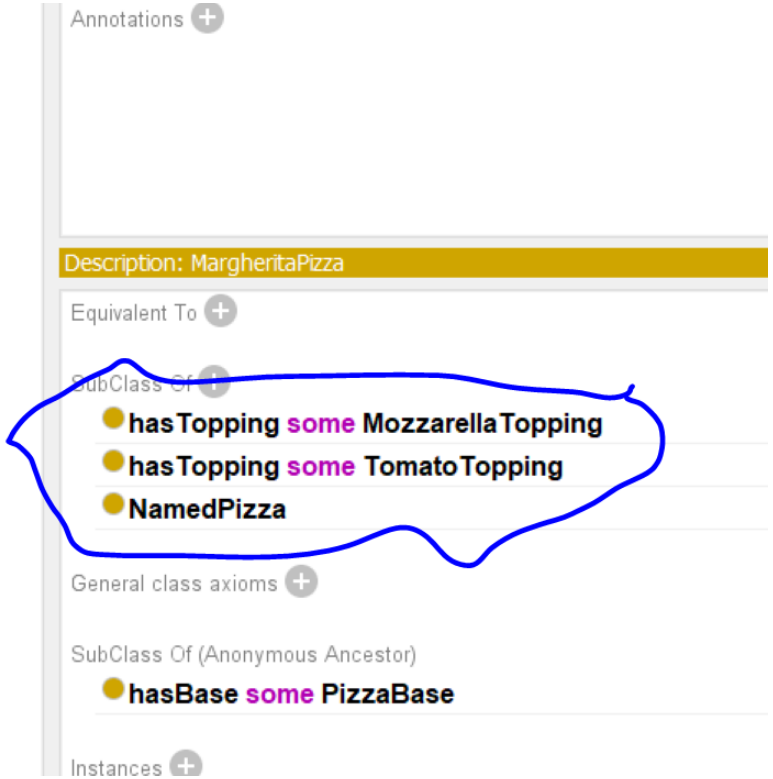




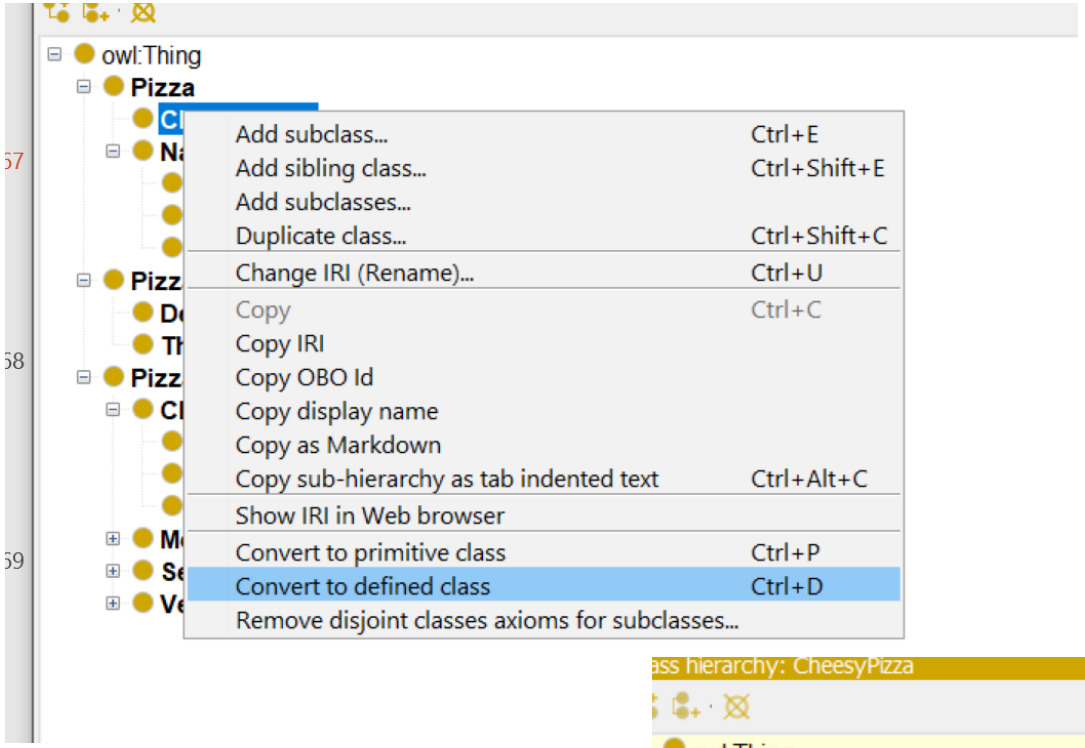
3. Margherita Pizza

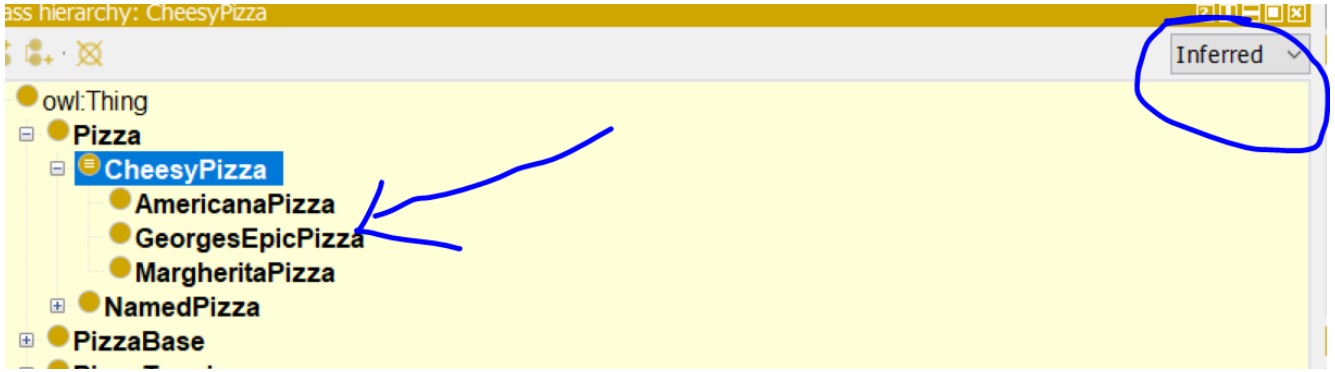
- Easy





4. Allow new inferences





- Comment!

Observe the difference for the range between object property range and datatype property

- there exist two property variants:
  - object properties
  - datatype properties

- **Object properties** have classes as range
  - `:author a owl:ObjectProperty .`

- Domain and Range of object properties

```
:author  a owl:ObjectProperty ;
         rdfs:domain :Book ;
         rdfs:range  :Writer .
```

$\exists author.\top \sqsubseteq Book$

$\top \sqsubseteq \forall author.Writer$

- **Datatype properties** have datatypes as range
  - `:publicationYear a owl:DatatypeProperty .`

- Domain and Range of datatype properties

```
:publicationYear a owl:DatatypeProperty ;
                 rdfs:domain :Book ;
                 rdfs:range  xsd:integer .
```

$\exists publicationYear.\top \sqsubseteq Book$

$\top \sqsubseteq \forall publicationYear.Integer$

Let’s discuss what we see

- Class

```
:Book a owl:Class .
:Writer a owl:Class .

:author  a owl:ObjectProperty ;
         rdfs:domain :Book ;
         rdfs:range  :Writer .

:publicationYear  a owl:DatatypeProperty ;
            rdfs:domain :Book ;
            rdfs:range  xsd:integer .

:GeorgeOrwell a Writer .
:NineteenEightyFour a :Book ;
          :author :GeorgeOrwell ;
          :publicationYear 1948 .
```