OWL Tutorial

adapted from

Presentation by the COODE and HyOntUse Projects

by

Photchanan Ratanajaipan

March 2009

OWL Tutorial : Overview

- Session 1: Interface basics
- Session 2: Defining a vegetarian pizza
- Session 3: Case Study

Session 1: Interface Basics

- Review: OWL Basics
- Intro: Protégé-OWL
- Interface: Creating Classes
- Concept: Disjointness
- Interface: Creating Properties
- Concept: Describing Classes
- Interface: Creating Restrictions

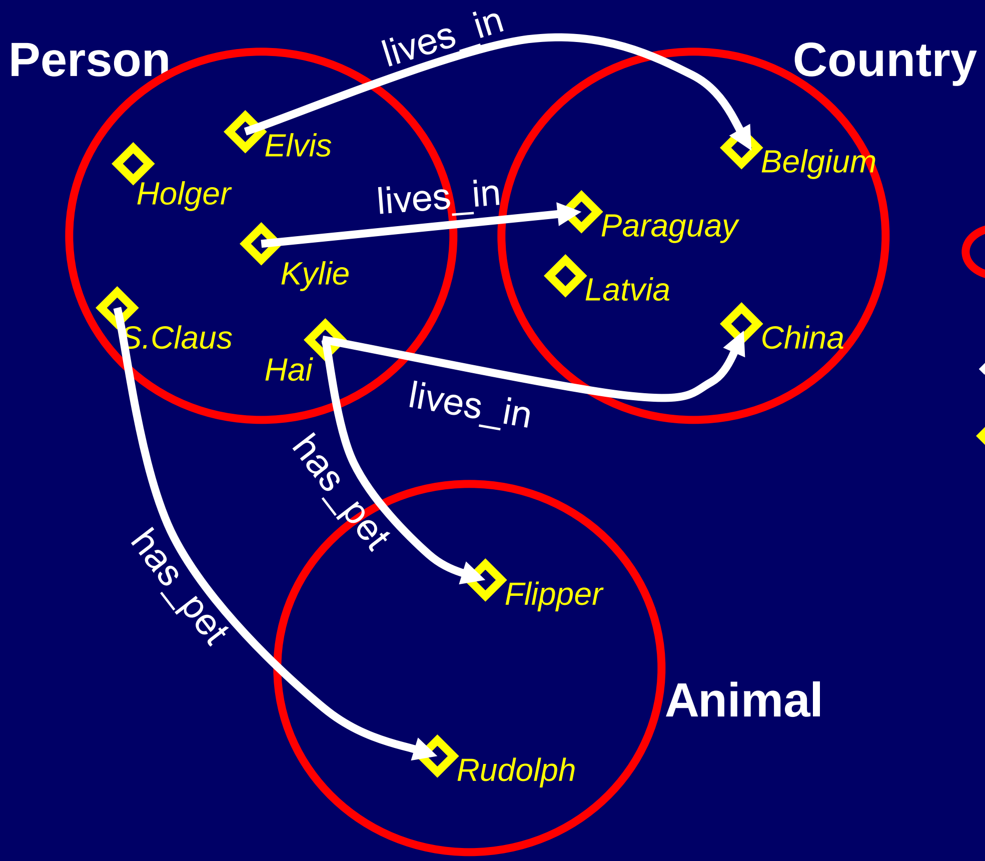
Review of OWL

OWL…

- is a W3C standard – Web Ontology Language
- comes in 3 flavours (lite, DL and full)
  - we are using OWL DL (Description Logic)
  - DL = decidable fragment of First Order Logic (FOL)
- is generally found in RDF/XML syntax
- is therefore not much fun to write by hand

So, we have tools to help us

OWL Constructs



○ = class (concept)

↗ = property (relationship)

◇ = individual (instance)

Get Protégé-OWL

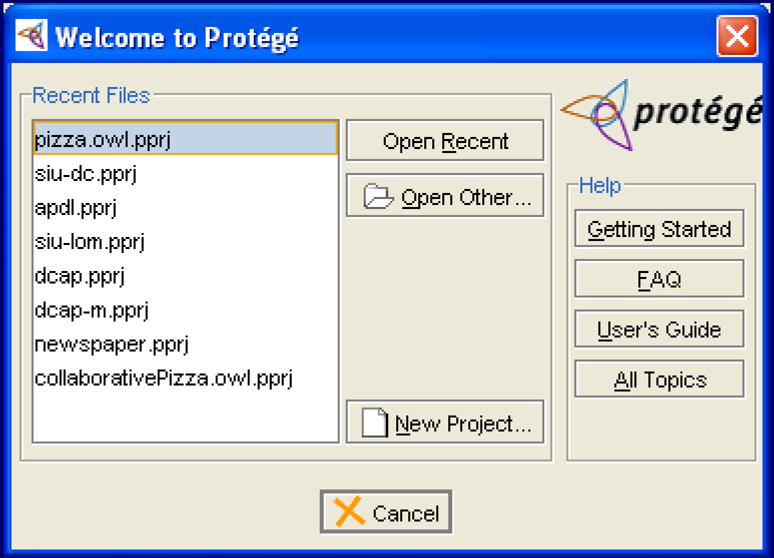
Logon to Windows

1. *Go to: http://protege.stanford.edu/download/registered.html*
2. *Download full Protégé 3.3.1 (current **released** version)*
3. *Install the software*

Starting Protégé-OWL

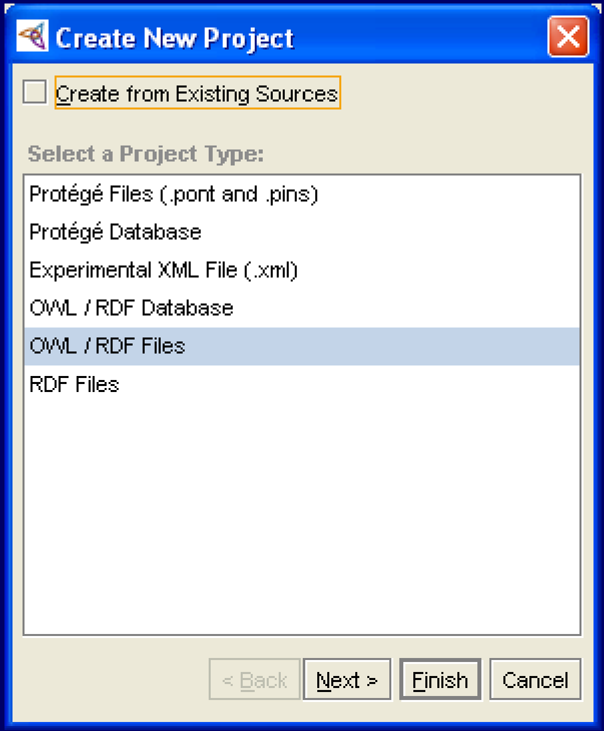
Run Protégé.exe

1. *Select “New Project…”*
2. *Select “OWL/RDF Files”*


Welcome to Protégé

Recent Files

- pizza.owl.pprj
- siu-dc.pprj
- apdl.pprj
- siu-lom.pprj
- dcap.pprj
- dcap-m.pprj
- newspaper.pprj
- collaborativePizza.owl.pprj

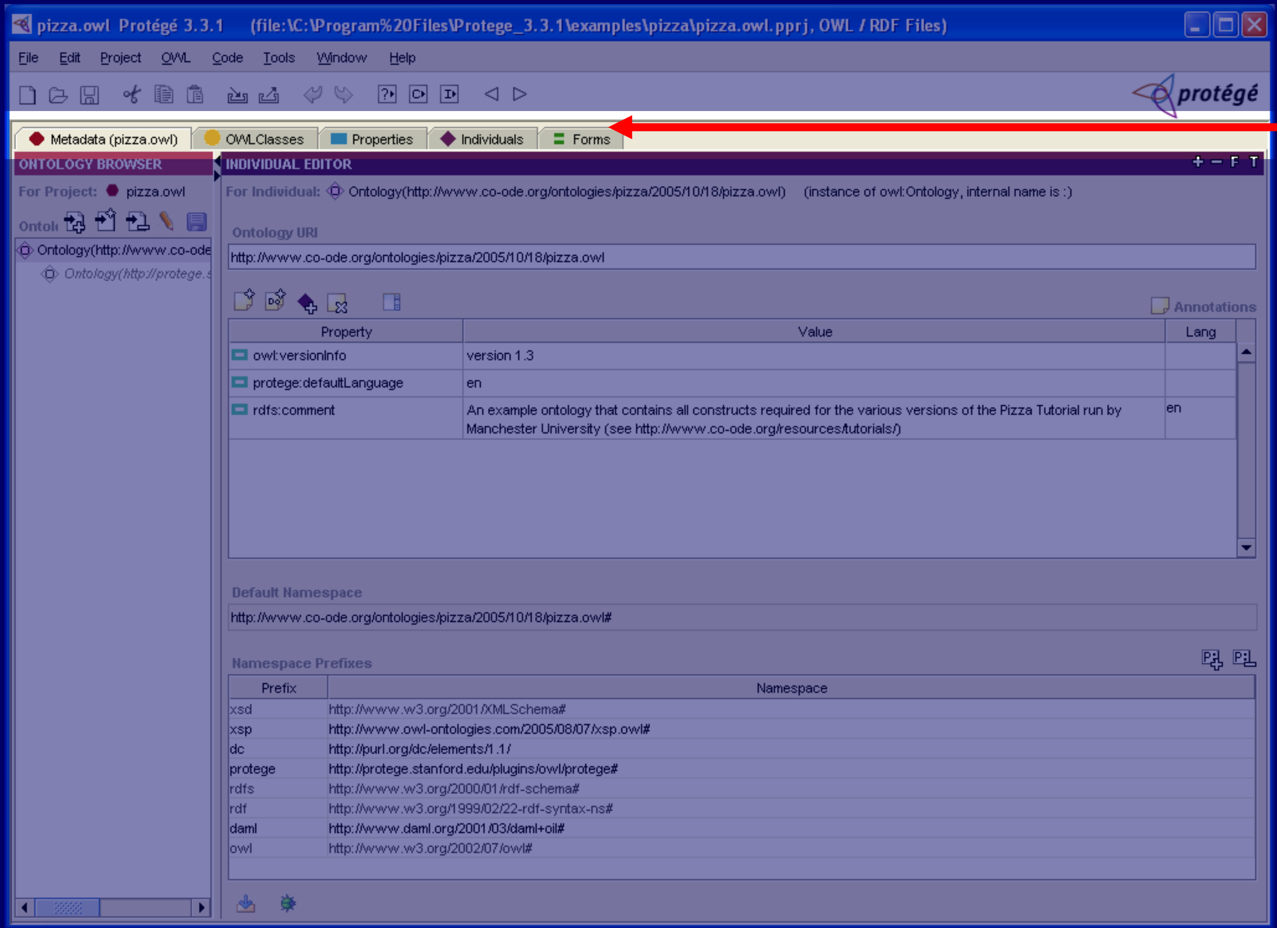

Create New Project

□ Create from Existing Sources

Select a Project Type:

- Protégé Files (.pont and .pins)
- Protégé Database
- Experimental XML File (.xml)
- OWL / RDF Database
- OWL / RDF Files
- RDF Files

Protégé OWL plugin



pizza.owl Protégé 3.3.1 (file:\C:\Program%20Files\Protege_3.3.1\examples\pizza\pizza.owl.pprj, OWL / RDF Files)

For Project: pizza.owl

Ontology(http://www.co-ode

Ontology(http://protege.s

For Individual: Ontology(http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl) (instance of owl:Ontology, internal name is :)

Ontology URI: http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl

Property	Value	Lang
owl:versionInfo	version 1.3	
protege:defaultLanguage	en	
rdfs:comment	An example ontology that contains all constructs required for the various versions of the Pizza Tutorial run by Manchester University (see http://www.co-ode.org/resources/tutorials/)	en

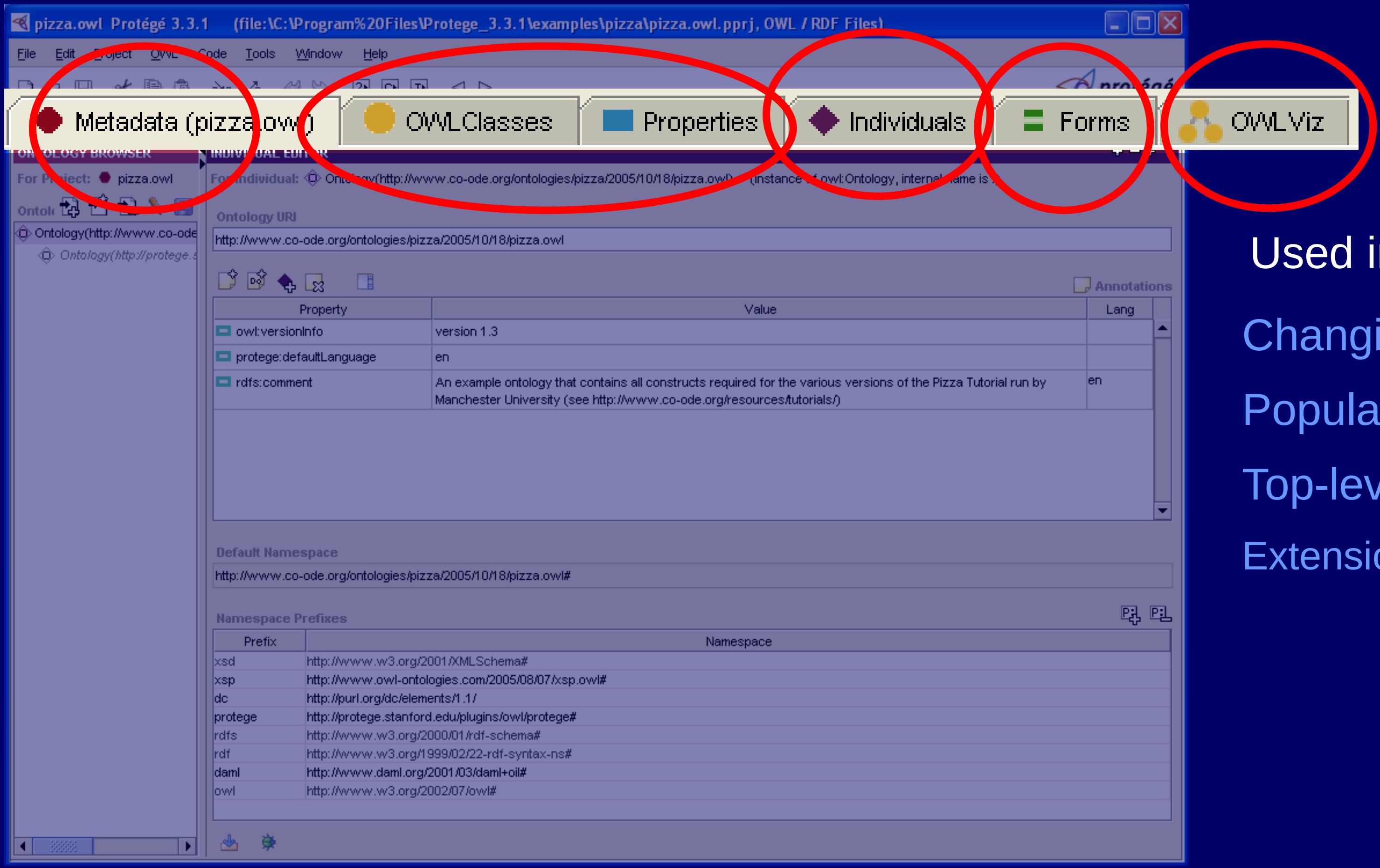
Default Namespace: http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl#

Namespace Prefixes

Prefix	Namespace
xsd	http://www.w3.org/2001/XMLSchema#
xsp	http://www.owl-ontologies.com/2005/08/07/xsp.owl#
dc	http://purl.org/dc/elements/1.1/
protege	http://protege.stanford.edu/plugins/owl/protege#
rdfs	http://www.w3.org/2000/01/rdf-schema#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
daml	http://www.daml.org/2001/03/daml+oil#
owl	http://www.w3.org/2002/07/owl#

Protégé tabs

Protégé OWL plugin: Tabs



pizza.owl Protégé 3.3.1 (file:\C:\Program%20Files\Protege_3.3.1\examples\pizza\pizza.owl.pprj, OWL / RDF Files)

For Project: pizza.owl

Ontology(http://www.co-ode

Ontology(http://protege.s

For Individual: Ontology(http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl) (instance of owl:Ontology, internal name is

Ontology URI

http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl

Property	Value	Lang
owl:versionInfo	version 1.3	
protege:defaultLanguage	en	
rdfs:comment	An example ontology that contains all constructs required for the various versions of the Pizza Tutorial run by Manchester University (see http://www.co-ode.org/resources/tutorials/)	en

Default Namespace

http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl#

Namespace Prefixes

Prefix	Namespace
xsd	http://www.w3.org/2001/XMLSchema#
xsp	http://www.owl-ontologies.com/2005/08/07/xsp.owl#
dc	http://purl.org/dc/elements/1.1/
protege	http://protege.stanford.edu/plugins/owl/protege#
rdfs	http://www.w3.org/2000/01/rdf-schema#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
daml	http://www.daml.org/2001/03/daml+oil#
owl	http://www.w3.org/2002/07/owl#

Used in this tutorial

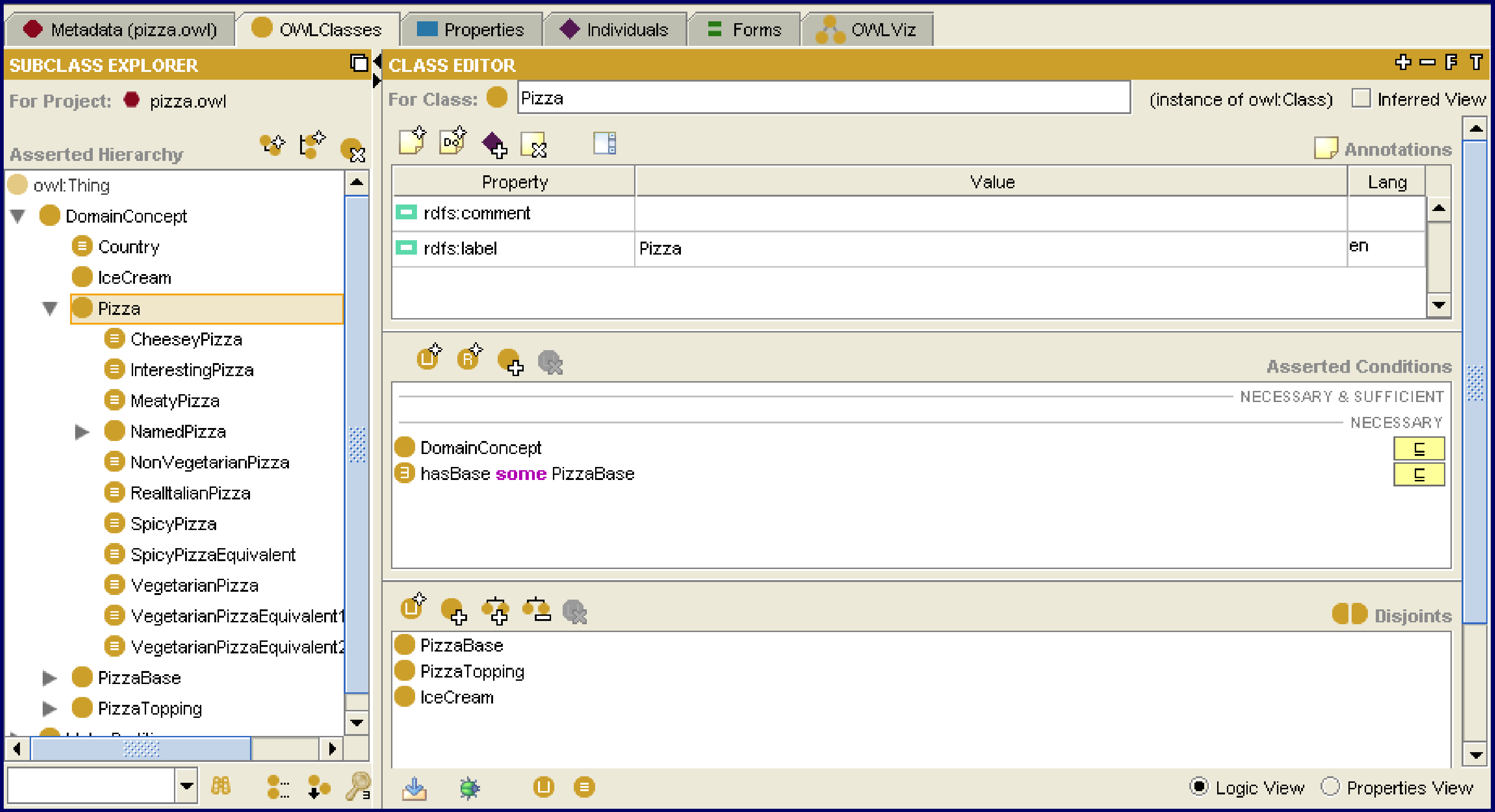
Changing the GUI

Populating the model

Top-level functionality

Extensions (visualisation)

Classes Tab



For Project: pizza.owl

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

For Class: Pizza (instance of owl:Class) □ Inferred View

Property	Value	Lang
rdfs:comment		
rdfs:label	Pizza	en

NECESSARY

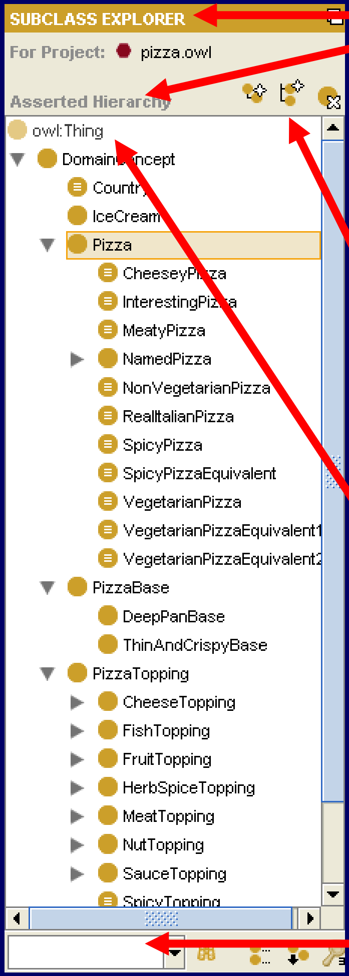
- DomainConcept ⊑
- hasBase **some** PizzaBase ⊑

Disjoints:

- PizzaBase
- PizzaTopping
- IceCream

● Logic View ○ Properties View

ClassesTab: Asserted Class Hierarchy

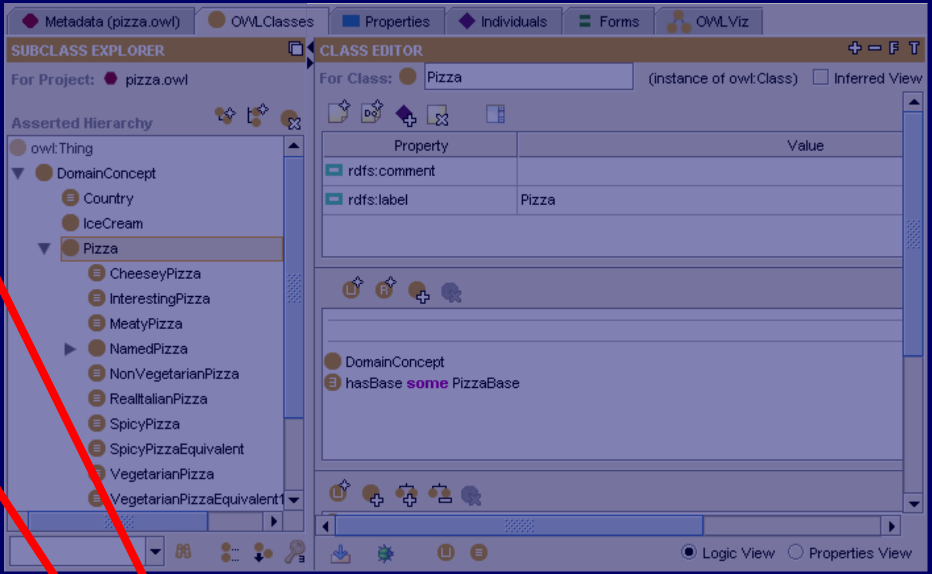


SUBCLASS EXPLORER

For Project: pizza.owl

Asserted Hierarchy

- owl:Thing
- DomainConcept
  - Country
  - IceCream
  - Pizza
    - CheeseyPizza
    - InterestingPizza
    - MeatyPizza
    - NamedPizza
    - NonVegetarianPizza
    - RealItalianPizza
    - SpicyPizza
    - SpicyPizzaEquivalent
    - VegetarianPizza
    - VegetarianPizzaEquivalent1
    - VegetarianPizzaEquivalent2
  - PizzaBase
    - DeepPanBase
    - ThinAndCrispyBase
  - PizzaTopping
    - CheeseTopping
    - FishTopping
    - FruitTopping
    - HerbSpiceTopping
    - MeatTopping
    - NutTopping
    - SauceTopping
    - SpicyTopping



For Project: pizza.owl

Asserted Hierarchy

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

For Class: Pizza (instance of owl:Class) □ Inferred View

Property	Value
rdfs:comment	
rdfs:label	Pizza

DomainConcept

hasBase **some** PizzaBase

● Logic View ○ Properties View

Subsumption hierarchy (superclass/subclass)

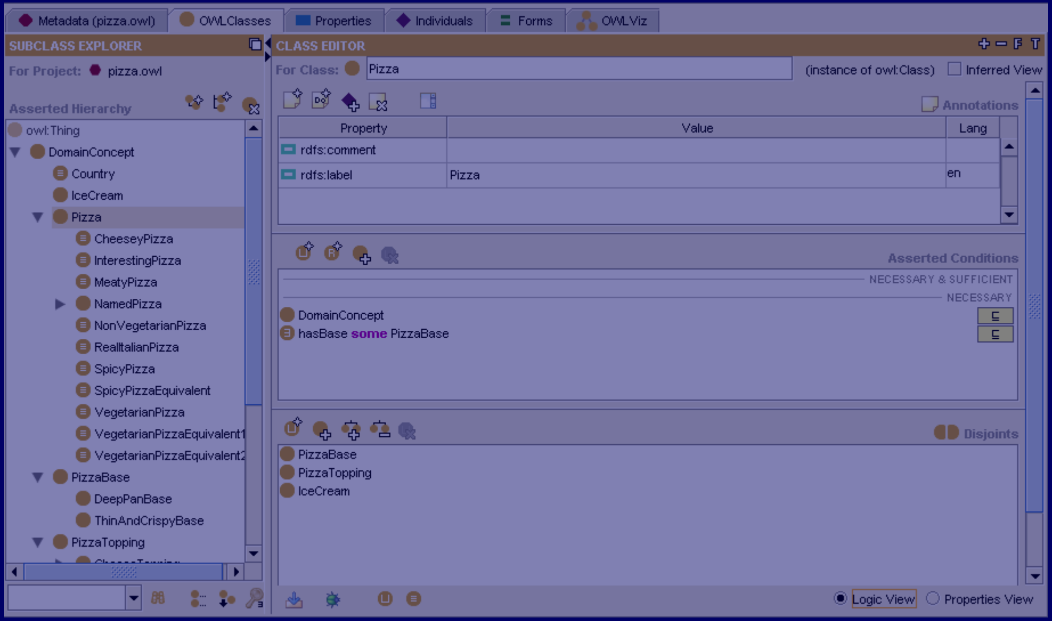
Structure as asserted by the ontology engineer

Create and Delete classes (actually subclasses!!)

Everything is a subclass of owl:Thing

Search for class

ClassesTab: Class Editor



For Class: Pizza (instance of owl:Class) □ Inferred View

Property	Value	Lang
rdfs:comment		
rdfs:label	Pizza	en

DomainConcept

hasBase **some** PizzaBase

PizzaBase

PizzaTopping

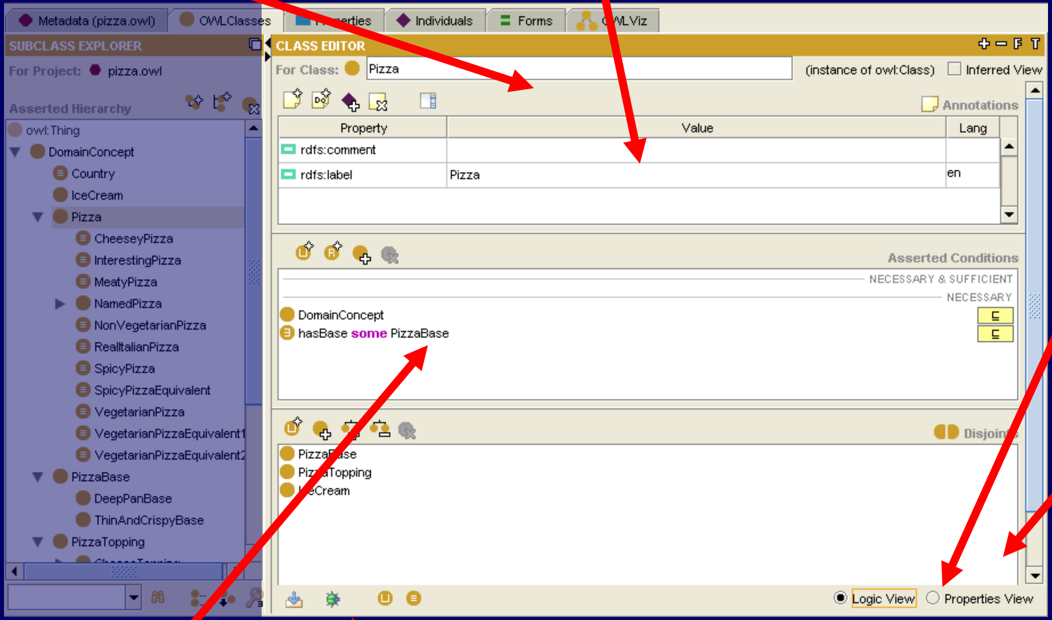
IceCream

● Logic View ○ Properties View

ClassesTab: Class Editor

Class annotations (for class metadata)

Class name and documentation



For Class: Pizza (instance of owl:Class) □ Inferred View

Property	Value	Lang
rdfs:comment		
rdfs:label	Pizza	en

DomainConcept ⊑
hasBase some PizzaBase ⊑

PizzaBase
PizzaTopping
IceCream

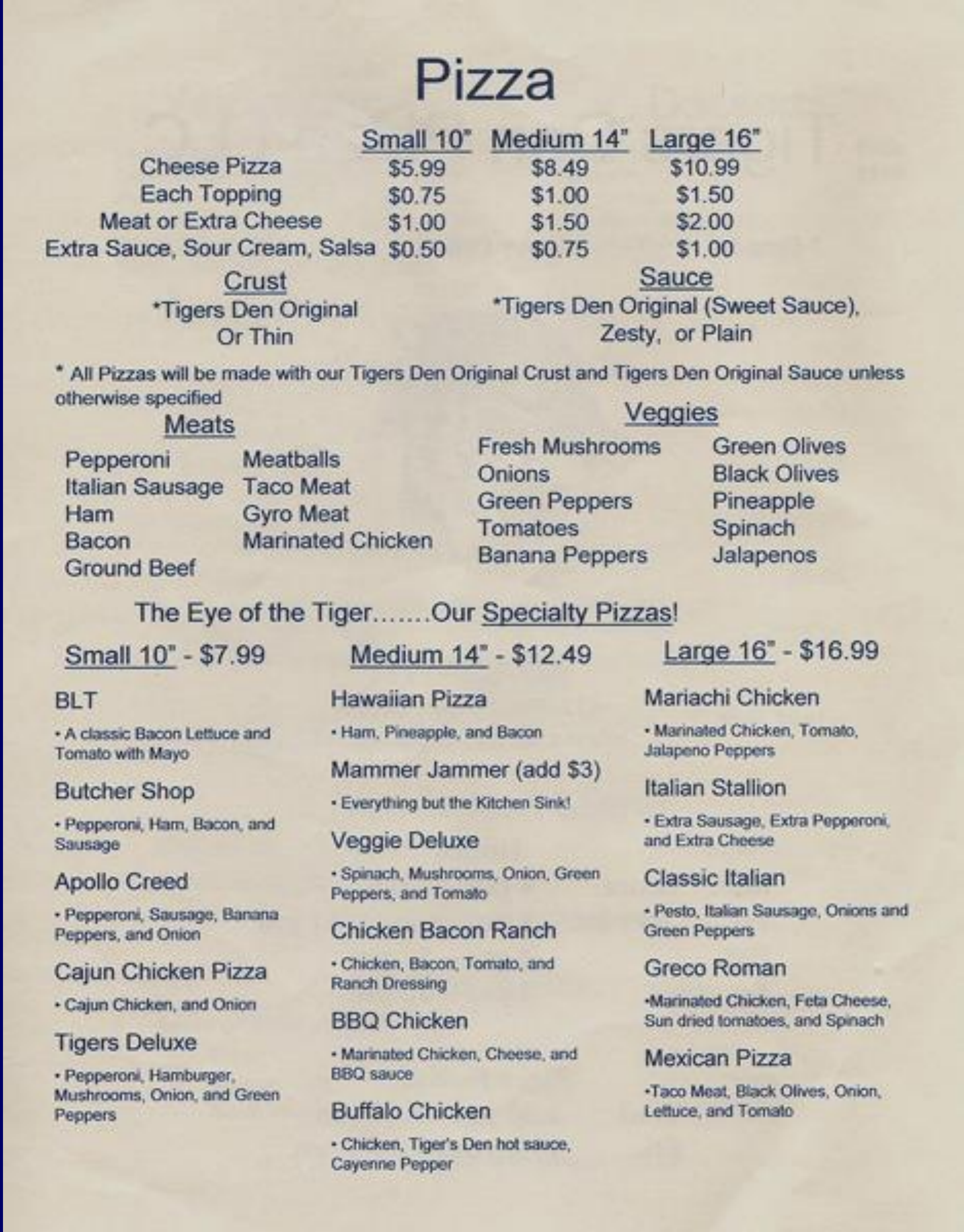
● Logic View ○ Properties View

Switch view to show Properties "available" to Class

Disjoints widget

Conditions Widget

Class-specific tools (find usage etc)



Pizza

	Small 10"	Medium 14"	Large 16"
Cheese Pizza	$5.99	$8.49	$10.99
Each Topping	$0.75	$1.00	$1.50
Meat or Extra Cheese	$1.00	$1.50	$2.00
Extra Sauce, Sour Cream, Salsa	$0.50	$0.75	$1.00

Crust

*Tigers Den Original
Or Thin

Sauce

*Tigers Den Original (Sweet Sauce),
Zesty, or Plain

* All Pizzas will be made with our Tigers Den Original Crust and Tigers Den Original Sauce unless otherwise specified

Meats

Pepperoni
Italian Sausage
Ham
Bacon
Ground Beef
Meatballs
Taco Meat
Gyro Meat
Marinated Chicken

Veggies

Fresh Mushrooms
Onions
Green Peppers
Tomatoes
Banana Peppers
Green Olives
Black Olives
Pineapple
Spinach
Jalapenos

The Eye of the Tiger.......Our Specialty Pizzas!

Small 10" - $7.99

BLT

- A classic Bacon Lettuce and Tomato with Mayo

Butcher Shop

- Pepperoni, Ham, Bacon, and Sausage

Apollo Creed

- Pepperoni, Sausage, Banana Peppers, and Onion

Cajun Chicken Pizza

- Cajun Chicken, and Onion

Tigers Deluxe

- Pepperoni, Hamburger, Mushrooms, Onion, and Green Peppers

Medium 14" - $12.49

Hawaiian Pizza

- Ham, Pineapple, and Bacon

Mammer Jammer (add $3)

- Everything but the Kitchen Sink!

Veggie Deluxe

- Spinach, Mushrooms, Onion, Green Peppers, and Tomato

Chicken Bacon Ranch

- Chicken, Bacon, Tomato, and Ranch Dressing

BBQ Chicken

- Marinated Chicken, Cheese, and BBQ sauce

Buffalo Chicken

- Chicken, Tiger's Den hot sauce, Cayenne Pepper

Large 16" - $16.99

Mariachi Chicken

- Marinated Chicken, Tomato, Jalapeno Peppers

Italian Stallion

- Extra Sausage, Extra Pepperoni, and Extra Cheese

Classic Italian

- Pesto, Italian Sausage, Onions and Green Peppers

Greco Roman

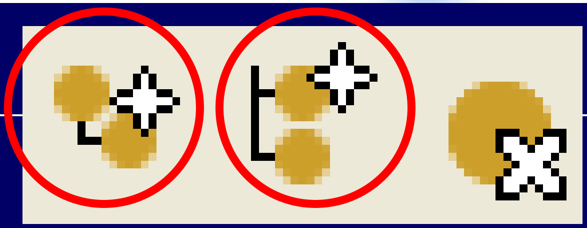
- Marinated Chicken, Feta Cheese, Sun dried tomatoes, and Spinach

Mexican Pizza

- Taco Meat, Black Olives, Onion, Lettuce, and Tomato

Create Classes

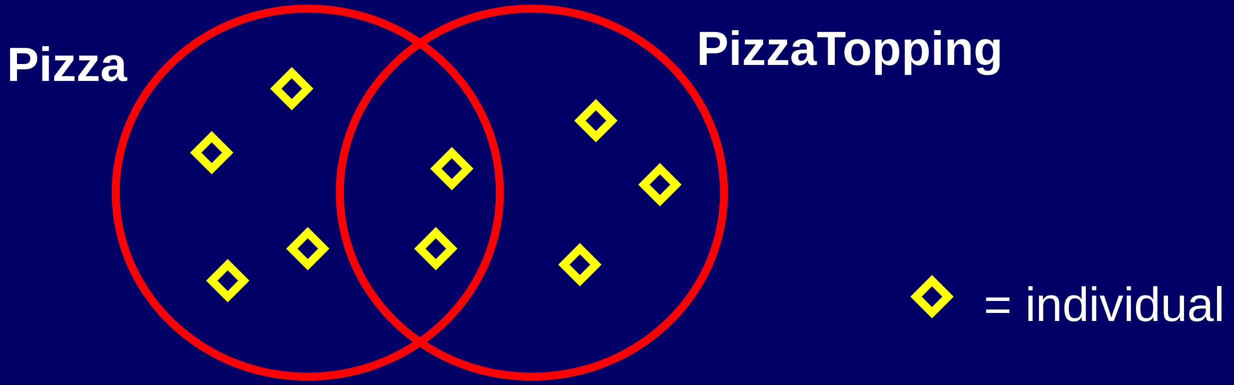
Start with your empty ontology



1. *Click the "Create subclass" button*
   (this is above the class hierarchy)
   A new class will be created as a subclass of ***owl:Thing***
2. *Type in a new name "DomainConcept" over the default*
   (press "enter" updates the hierarchy)
3. *Req. for later labs: document your class using the rdfs:comment field*
4. *Create another class called "Pizza" by clicking the "Create sibling class"*
   You will notice that ***Pizza*** *has been created as a subclass of* ***DomainConcept*** *as this was the class selected when the button was pressed. You can also right-click any class and select "Create Class"*
5. *Create two more subclasses of* ***DomainConcept*** *called "PizzaTopping" and "PizzaBase".*
   Any mistakes, use the "Delete Class" button next to "Create Class"

Disjointness

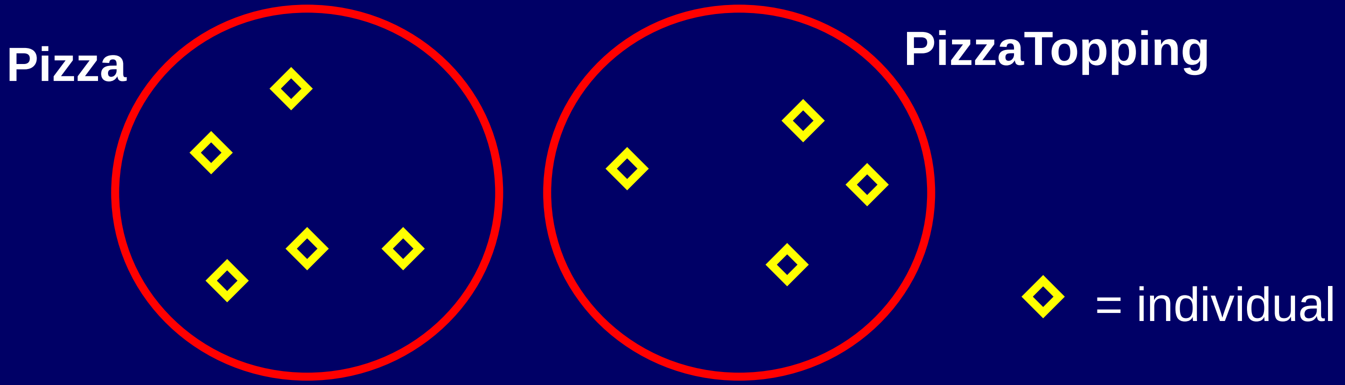
- OWL assumes that classes overlap



- This means an individual could be both a **Pizza** and a **PizzaTopping** at the same time
- We want to state this is not the case

Disjointness

- If we state that classes are disjoint



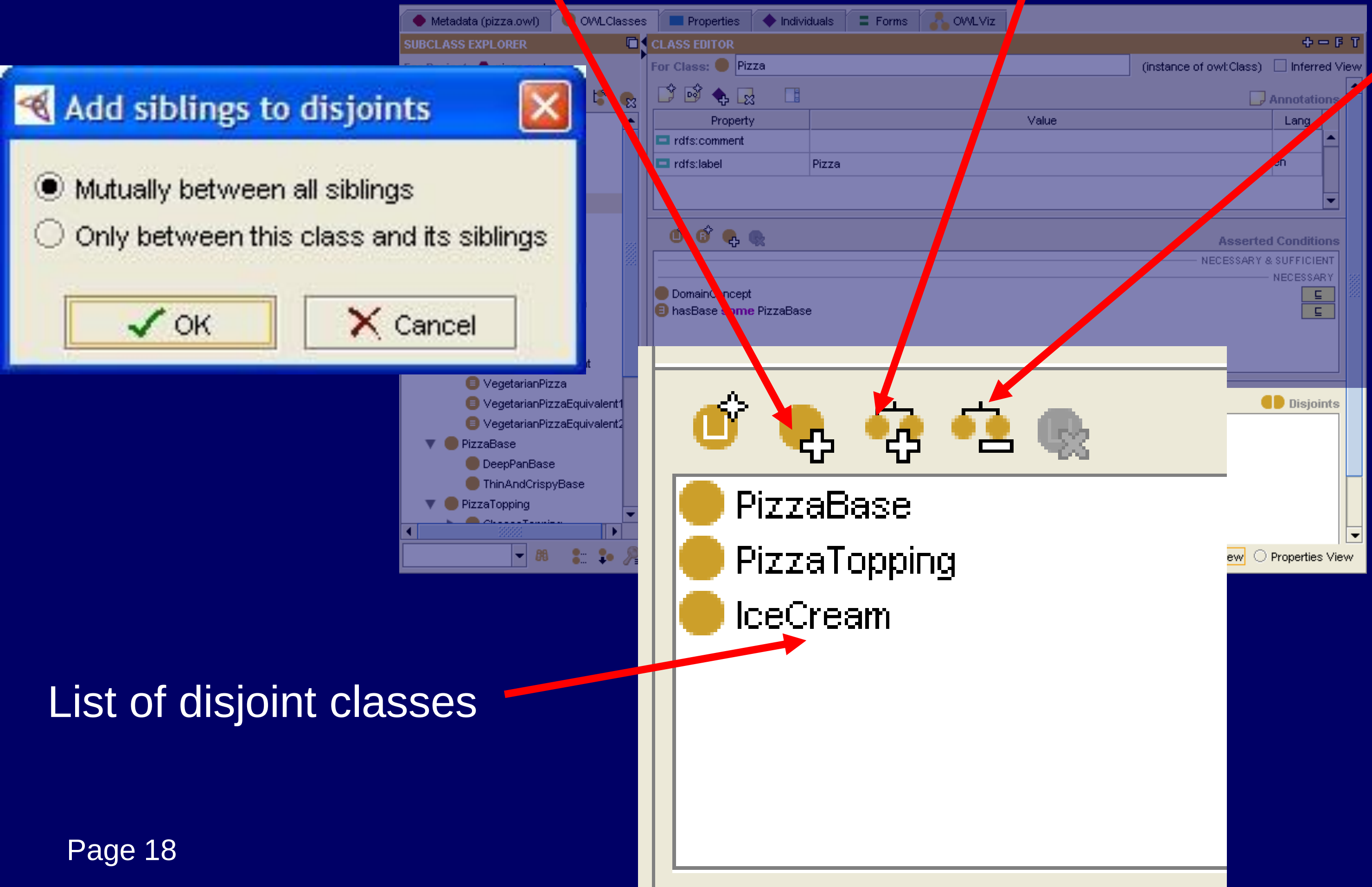
- This means an individual cannot be both a **Pizza** and a **PizzaTopping** at the same time
- We must do this explicitly in the interface

ClassesTab: Disjoints Widget

Add siblings as disjoint

Add new disjoint

Remove disjoint siblings



Add siblings to disjoints

● Mutually between all siblings

○ Only between this class and its siblings

✓ OK ✗ Cancel

PizzaBase

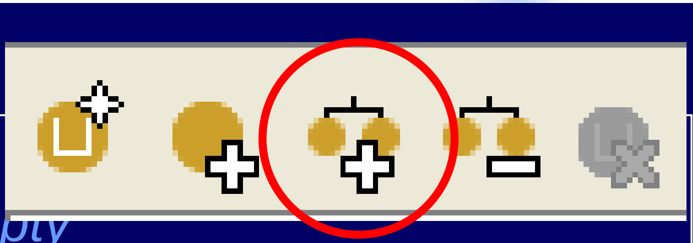
PizzaTopping

IceCream

List of disjoint classes

Make Classes Disjoint

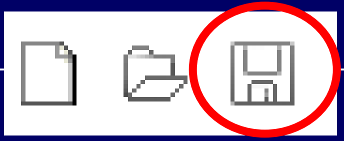
Start with your existing ontology



1. *Select the* **Pizza** class
   You will notice that the disjoints widget is empty
2. *Click the “Add all siblings…” button*
   The “Add siblings to disjoints dialog pops up
3. *Select the “Mutually between all siblings” option and OK*
   PizzaTopping *and* **PizzaBase** *appear in the disjoints widget*
4. *Select the* **PizzaTopping** *class*
   Pizza *and* **PizzaBase** *are already in the disjoints widget*
5. *Note that the same applies for* **PizzaBase**

Save Your Work

OWL = easy to make mistakes – save regularly



1. *Select File → Save*

   A dialog (as shown) will pop up

2. *Select a file using a file selector by clicking the button on the top right*

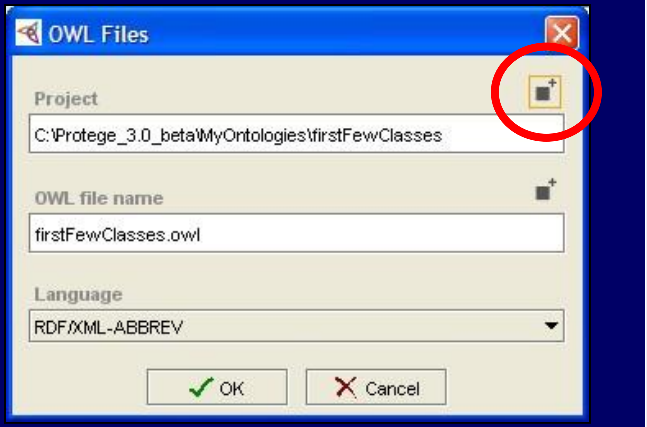
   You will notice that there are 2 files created

   .pprj – the project file

   this just stores information about the GUI and the workspace

   .owl – the OWL file

   this is where your ontology is stored in RDF/OWL format



Project
C:\Protege_3.0_beta\MyOntologies\firstFewClasses

OWL file name
firstFewClasses.owl

Language
RDF/XML-ABBREV

3. *Select OK*

Create PizzaToppings

Start with your existing ontology

1. *Create subclasses of* **PizzaTopping:**
   - **CheeseTopping**
   - **VegetableTopping**
   - **MeatTopping**
2. *Make these subclasses all disjoint from one another*
   (remember to chose "Mutually between all siblings" when prompted)
3. *Create subclasses of* **CheeseTopping:**
   MozzarellaTopping, **ParmesanTopping**
4. *Make these subclasses all disjoint from one another*
5. *Create subclasses of* **VegetableTopping** *and make them disjoint:*
   TomatoTopping, **MushroomTopping**
6. *Save to another file using File → Save As…*

What have we got?

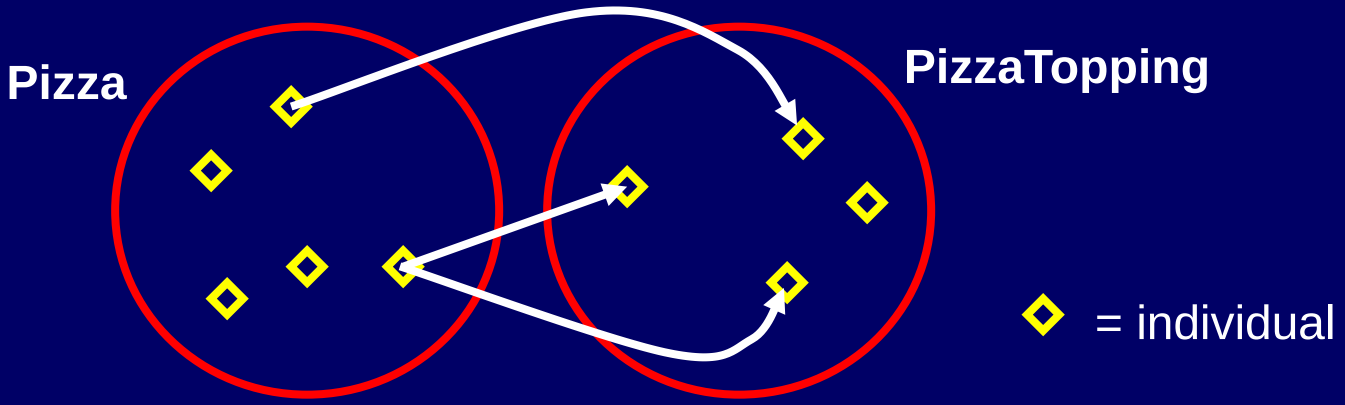
- We’ve created a tree of disjoint classes
- Disjoints are inherited down the tree

  e.g. something that is a **TomatoTopping** cannot be a **Pizza** because its superclass, **PizzaTopping**, is disjoint from **Pizza**

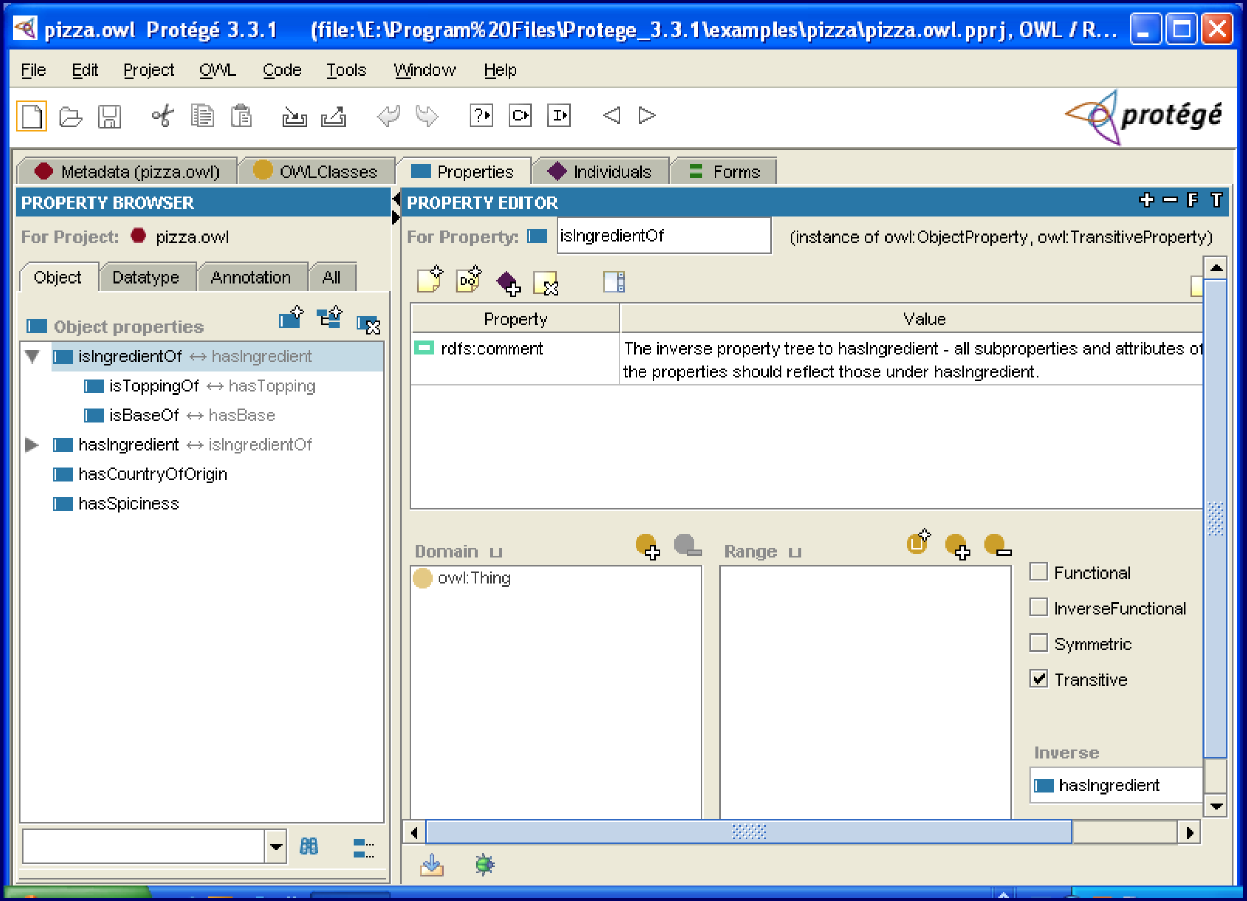
- You should now be able to select every class (except **DomainConcept**) and see its siblings in the disjoints widget

What are we missing?

- This is not a semantically rich model
- Apart from “is kind of” and “is not kind of”, we currently don’t have any other information of interest
- We want to say more about **Pizza** individuals, such as their relationship with other individuals
- We can do this with properties



Properties Tab



pizza.owl Protégé 3.3.1 (file:\E:\Program%20Files\Protege_3.3.1\examples\pizza\pizza.owl.pprj, OWL / R...

For Project: pizza.owl

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

For Property: isIngredientOf (instance of owl:ObjectProperty, owl:TransitiveProperty)

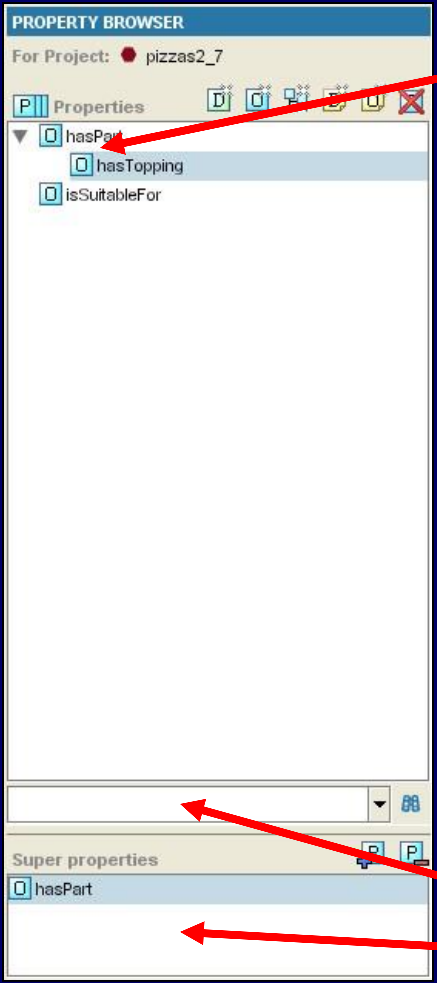
Property	Value
rdfs:comment	The inverse property tree to hasIngredient - all subproperties and attributes of the properties should reflect those under hasIngredient.

Domain: owl:Thing

□ Functional
□ InverseFunctional
□ Symmetric
☑ Transitive

Inverse: hasIngredient

Properties Tab: Property Browser

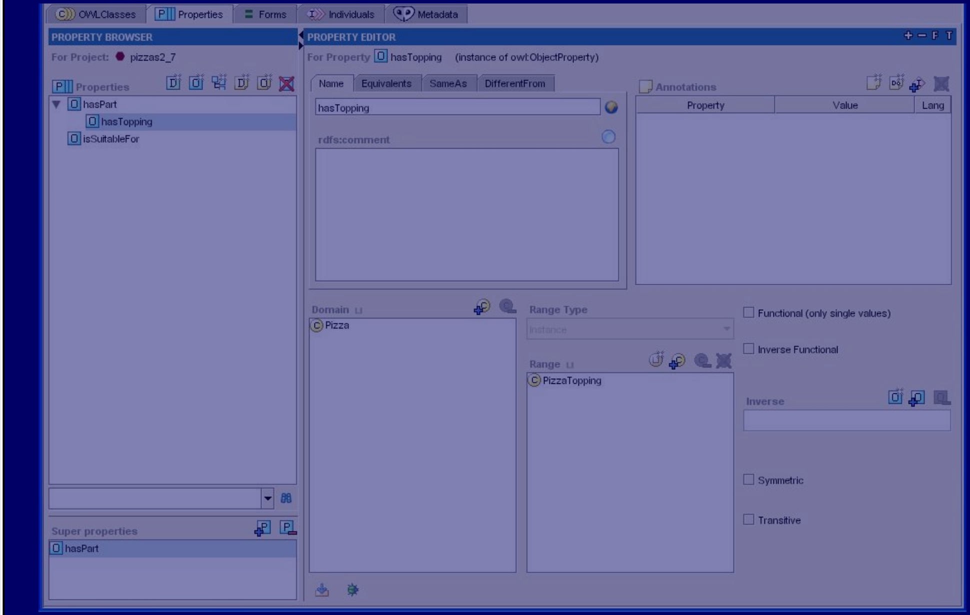


For Project: ● pizzas2_7

- hasPart
  - hasTopping
- isSuitableFor

Super properties: hasPart

Properties can be in a hierarchy



For Project: ● pizzas2_7

- hasPart
  - hasTopping
- isSuitableFor

For Property hasTopping (instance of owl:ObjectProperty)

Name: hasTopping

Domain: Pizza

Range Type: Instance

Range: PizzaTopping

□ Functional (only single values)

□ Inverse Functional

□ Symmetric

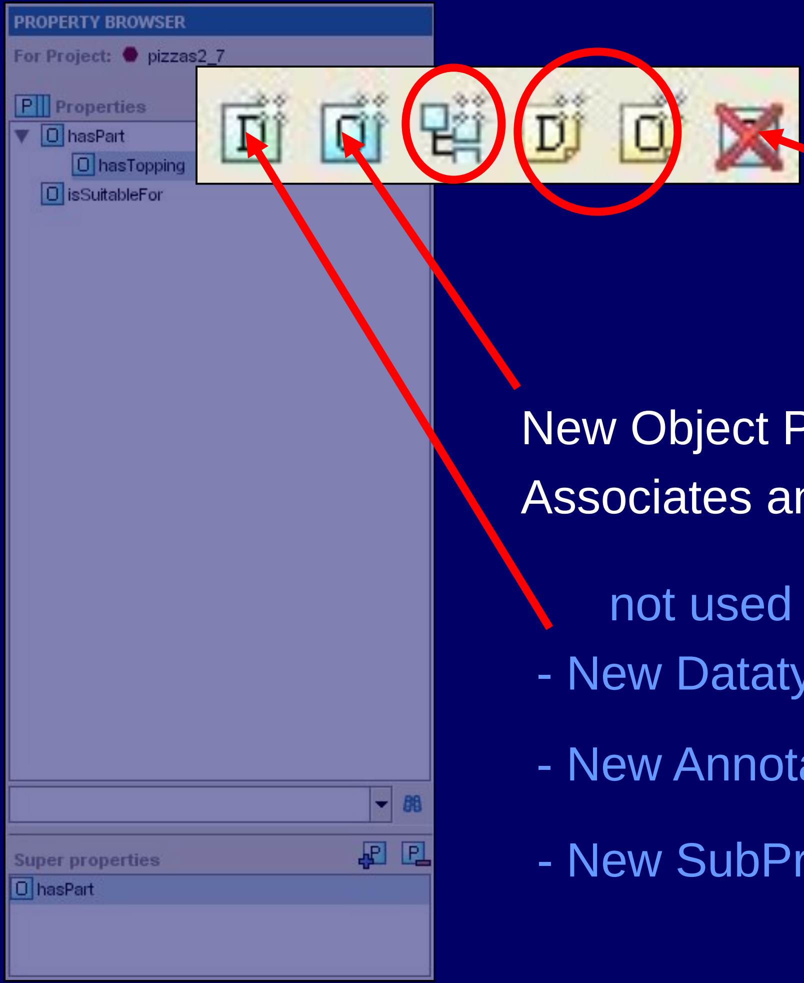
□ Transitive

Super properties: hasPart

Search for property

SuperProperties of the current selected

Properties Tab: Property Browser



Delete Property

New Object Property:
Associates an individual to another individual

not used today:

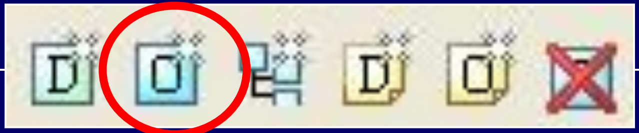
- New Datatype Property (String, int etc)

- New Annotation Properties for metadata

- New SubProperty – ie create “under” the current selection

Create a Property

Start with your existing ontology



1. *Switch to the Properties tab*
   There are currently no properties, so the list is blank
2. *Create a new Object property using the button in the property browser*
3. *Call the new Property "hasTopping"*
4. *Create another Object Property called "hasBase"*
5. *Save under a new filename*

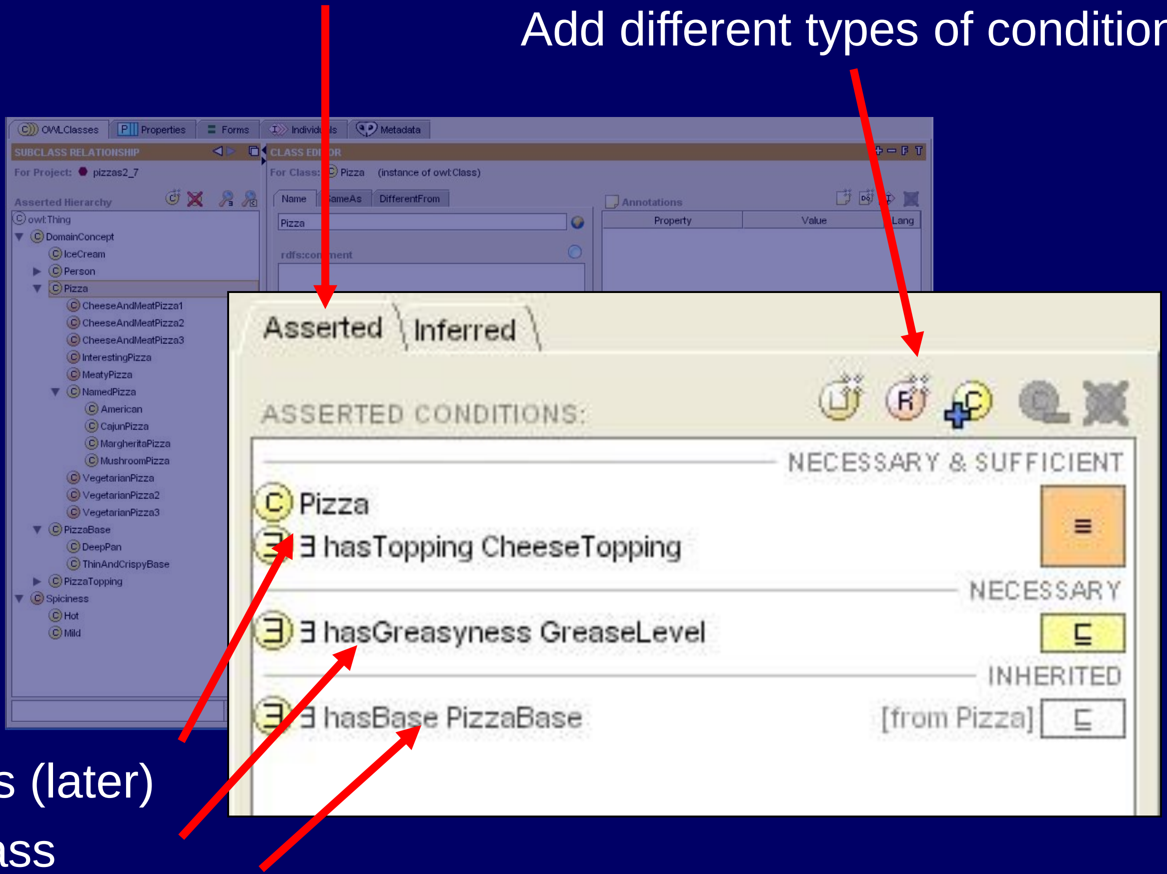
Associating Properties with Classes

- We now have two properties we want to use to describe **Pizza** individuals.
- To do this, we must go back to the **Pizza** class and add some further information
- This comes in the form of Restrictions (which are a type of Condition)

ClassesTab: Conditions Widget

Conditions asserted by the ontology engineer

Add different types of condition



For Project: pizzas2_7

For Class: Pizza (instance of owl:Class)

Pizza

- owl:Thing
- DomainConcept
  - IceCream
  - Person
  - Pizza
    - CheeseAndMeatPizza1
    - CheeseAndMeatPizza2
    - CheeseAndMeatPizza3
    - InterestingPizza
    - MeatyPizza
    - NamedPizza
      - American
      - CajunPizza
      - MargheritaPizza
      - MushroomPizza
    - VegetarianPizza
    - VegetarianPizza2
    - VegetarianPizza3
  - PizzaBase
    - DeepPan
    - ThinAndCrispyBase
  - PizzaTopping
- Spiciness
  - Hot
  - Mild

NECESSARY & SUFFICIENT

Pizza

∃ hasTopping CheeseTopping ≡

NECESSARY

∃ hasGreasyness GreaseLevel ⊑

INHERITED

∃ hasBase PizzaBase [from Pizza] ⊑

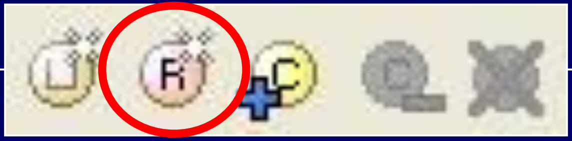
Definition of the class (later)

Description of the class

Conditions inherited from superclasses

Create a Restriction

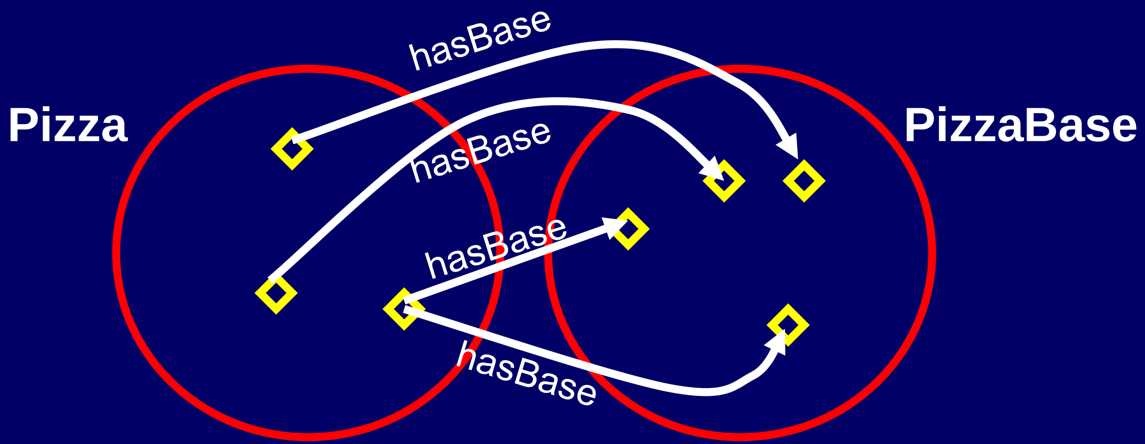
Start with your existing ontology



1. *Switch to the OWL Classes tab*
2. *Select* **Pizza**
   Notice that the conditions widget only contains one item, ***DomainConcept*** *with a Class icon.*
   Superclasses show up in the conditions widget in this way
3. *Click the "Create Restriction" button*
   A dialog pops up that we will investigate in a minute
4. *Select "hasBase" from the Restricted Property pane*
5. *Leave the Restriction type as "someValuesFrom"*
6. *Type "PizzaBase" in the Filler expression editor, then Click OK*
   A restriction has been added to the Conditions widget

What does this mean?

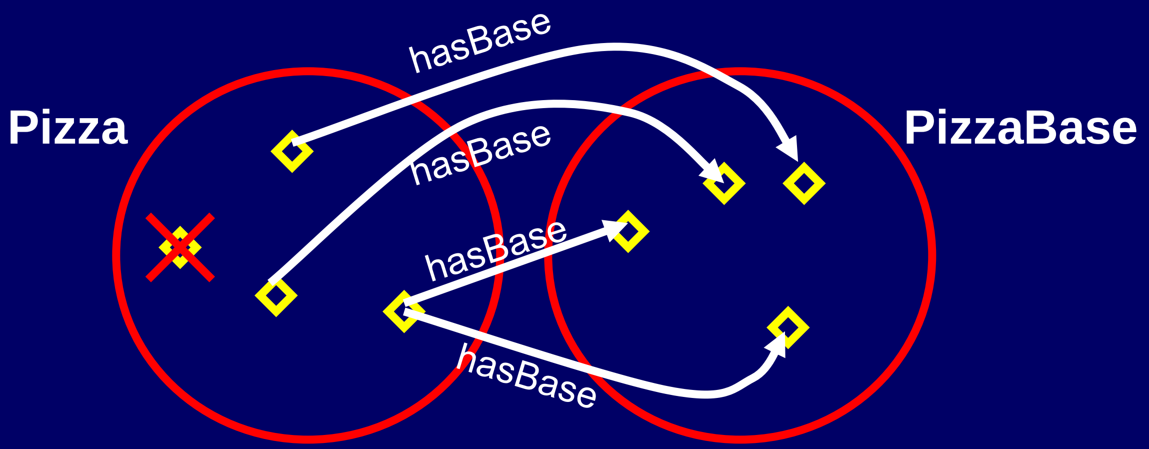
- We have created a restriction: ∃ hasBase **PizzaBase** on Class **Pizza** as a necessary condition



- “If an individual is a member of this class, it is necessary that it has at least one hasBase relationship with an individual from the class **PizzaBase**”
- “Every individual of the **Pizza** class must have at least one base from the class **PizzaBase**”

What does this mean?

- We have created a restriction: ∃ hasBase **PizzaBase** on Class **Pizza** as a necessary condition



- “There can be no individual, that is a member of this class, that does not have at least one hasBase relationship with an individual from the class **PizzaBase**”

Restrictions Popup

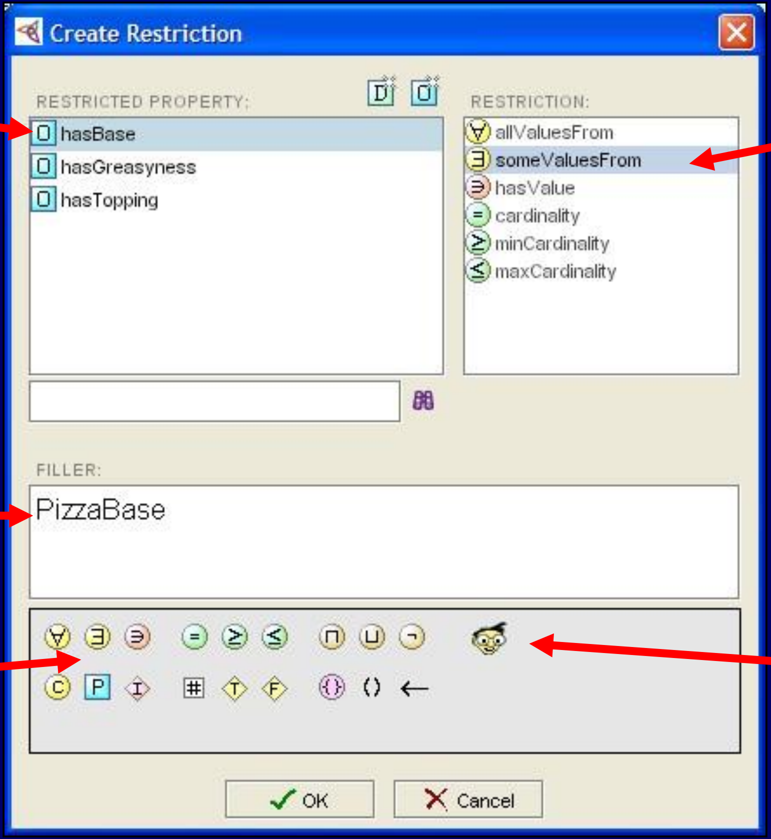
Restricted Property

Restriction Type

Filler Expression

Expression Construct Palette

Syntax check



Create Restriction

RESTRICTED PROPERTY:

- hasBase
- hasGreasyness
- hasTopping

RESTRICTION:

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

FILLER:

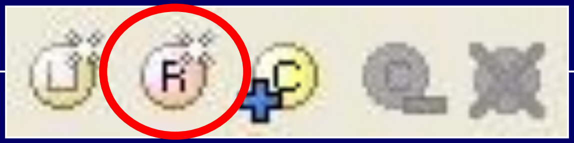
PizzaBase

Restriction Types

∃	Existential, someValuesFrom	“Some”, “At least one”
∀	Universal, allValuesFrom	“Only”
∋	hasValue	“equals x”
=	Cardinality	“Exactly n”
≤	Max Cardinality	“At most n”
≥	Min Cardinality	“At least n”

Another Existential Restriction

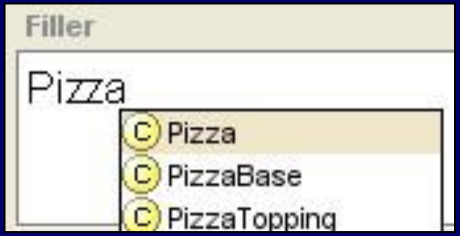
Start with your existing ontology



1. *Make sure* ***Pizza*** *is selected*
2. *Create a new Existential (SomeValuesFrom) Restriction with the hasTopping property and a filler of* ***PizzaTopping***

When entering the filler, you have 2 shortcut methods rather than typing the entire classname:

1) enter a partial name and use Tab to autocomplete

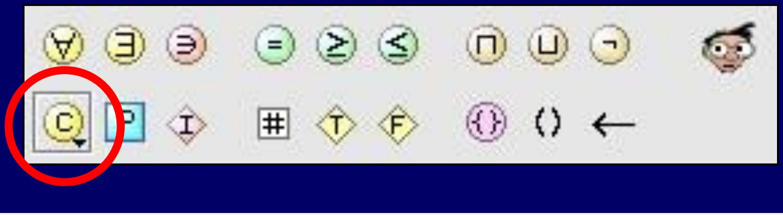


Filler

Pizza

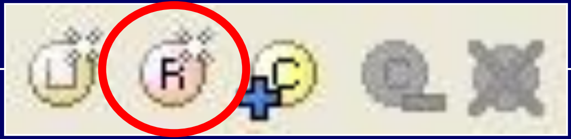
- Pizza
- PizzaBase
- PizzaTopping

2) use the select Class button on the editor palette



Create a Universal Restriction

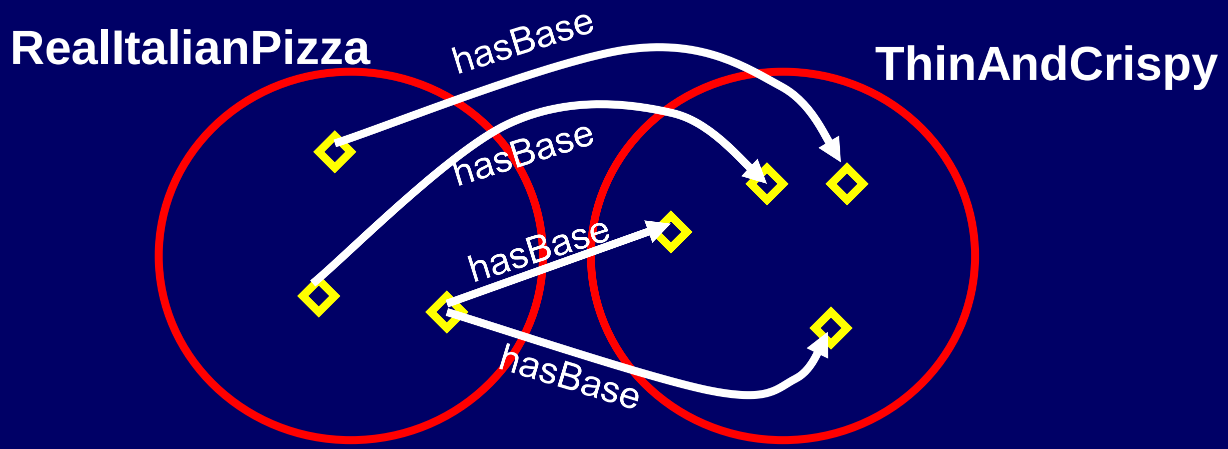
Start with your existing ontology



1. *Create 2 disjoint subclasses of* **PizzaBase** *called "ThinAndCrispy" and "DeepPan"*
2. *Create a subclass of* **Pizza** *called "RealItalianPizza"*
3. *Create a new Universal (AllValuesFrom) Restriction on* **RealItalianPizza** *with the hasBase property and a filler of* **ThinAndCrispy**

What does this mean?

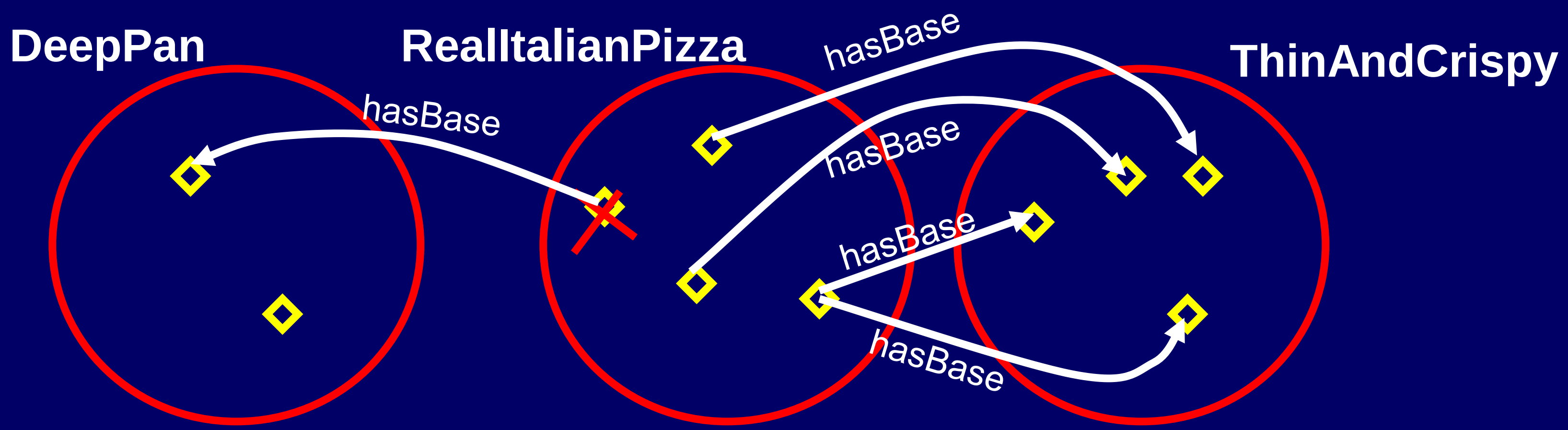
- We have created a restriction: ∀ hasBase **ThinAndCrispy** on Class **RealItalianPizza** as a necessary condition



- "If an individual is a member of this class, it is necessary that it must only have a hasBase relationship with an individual from the class **ThinAndCrispy**"

What does this mean?

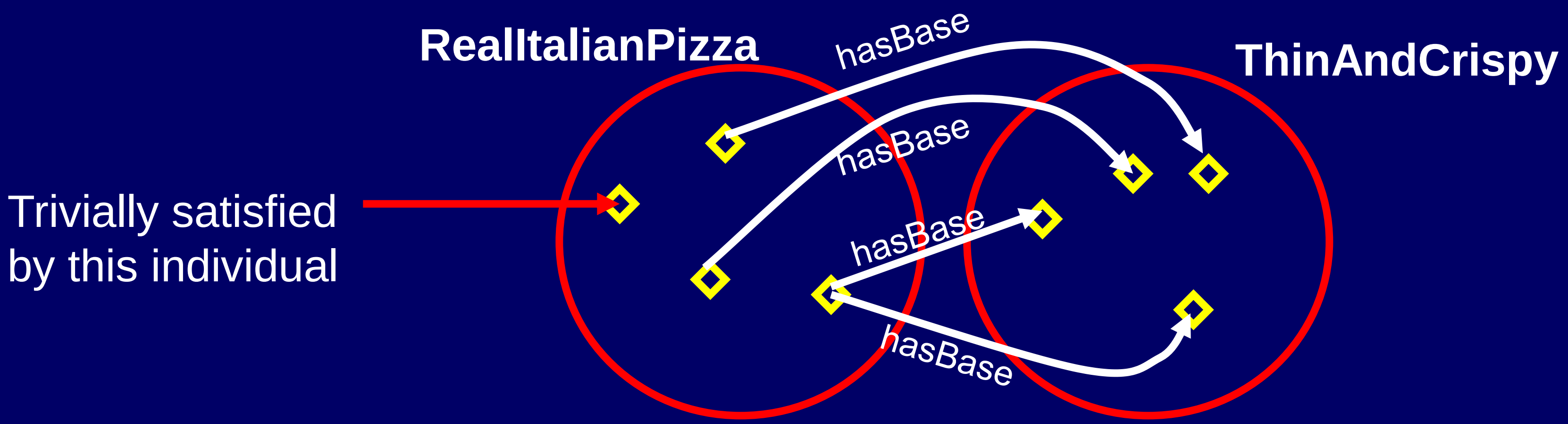
- We have created a restriction: ∀ hasBase **ThinAndCrispy** on Class **RealItalianPizza** as a necessary condition



- "No individual of the **RealItalianPizza** class can have a base from a class other than **ThinAndCrispy**"

Universal Warning – Trivial Satisfaction

- If we had not already inherited: ∃ hasBase **PizzaBase** from Class **Pizza** the following could hold



- “If an individual is a member of this class, it is necessary that it must only have a hasBase relationship with an individual from the class **ThinAndCrispy,** or no hasBase relationship at all”
- ie Universal Restrictions by themselves do not state “at least one”

Summary

You should now be able to:

- identify components of the Protégé-OWL Interface
- create Primitive Classes
- create Properties
- create some basic Restrictions on a Class using Existential and Universal qualifiers

More exercises: Create a MargheritaPizza

Start with your existing ontology

1. *Create a subclass of* **Pizza** *called* **NamedPizza**
2. *Create a subclass of* **NamedPizza** *called* **MargheritaPizza**
3. *Create a restriction to say that:*
   “Every MargheritaPizza must have at least one topping from TomatoTopping”
4. *Create another restriction to say that:*
   “Every MargheritaPizza must have at least one topping from MozzarellaTopping”

More exercises:
Create other pizzas

Start with your existing ontology

1. *Add more topping ingredients as subclasses of PizzaTopping*
   Use the hierarchy, but be aware of disjoints
2. *Create more subclasses of* **NamedPizza**
3. *Create a restrictions on these pizzas to describe their ingredients*
4. *Save this for the next session*

OWL Tutorial: Session II

adapted from

Presentation by the COODE and HyOntUse Projects

by

Photchanan Ratanajaipan

OWL Tutorial : Overview

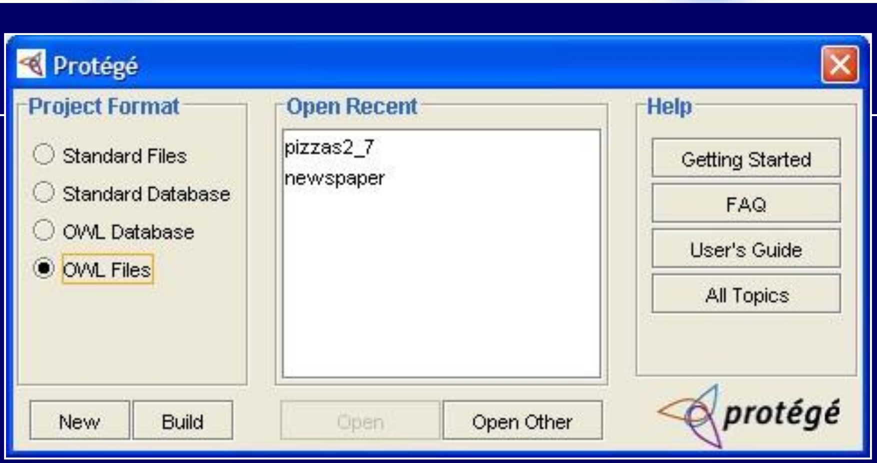
- Session 1: Interface basics
- Session 2: Defining a vegetarian pizza

Session 2: Vegetarian Pizza

- Issue: Primitive Classes & Polyhierarchies
- Advanced: Reasoning
- Advanced: Creating Defined Classes
- Union Classes: Covering Axioms
- Example: Creating a Vegetarian Pizza
- Issue: Open World Assumption
- Union Classes: Closure

Loading OWL files from scratch

Run Protégé.exe



1. *If you've only got an OWL file:*
   Select "OWL Files" as the Project Format, then "Build" to select the .owl file

2. *If you've got a valid project file*:*
   Select "OWL Files" as the Project Format, and then "Open Other" to find the .pprj file (if you've already opened it, it will be in "Open Recent")

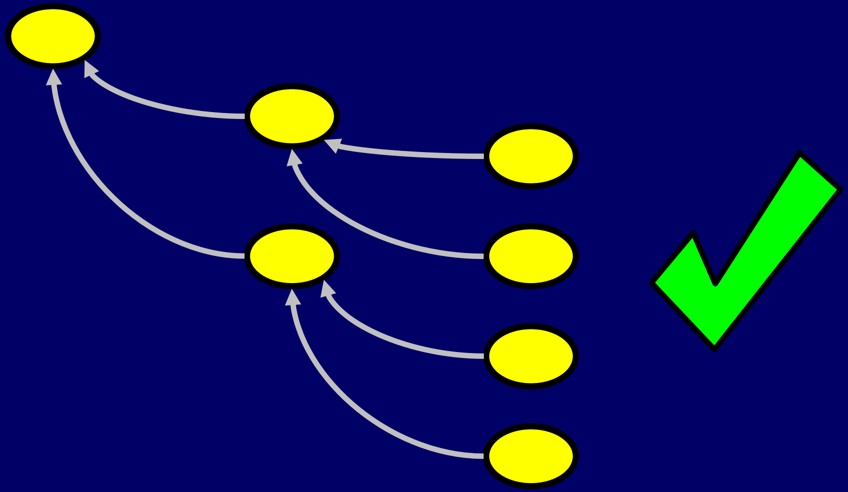
3. *Open C:\Protégé_3.0_beta\examples\pizzas\pizzas2_0.owl*

** ie one created on this version of Protégé - the s/w gets updated once every few days, so don't count on it unless you've created it recently– safest to build from the .owl file if in doubt*

Primitive Classes

- All classes in our ontology so far are Primitive
- We describe primitive pizzas
- Primitive Class = only Necessary Conditions
- They are marked as yellow in the class hierarchy

We condone building a disjoint tree of primitive classes



Describing Primitive Pizza Classes

Start with pizzas2_0.owl

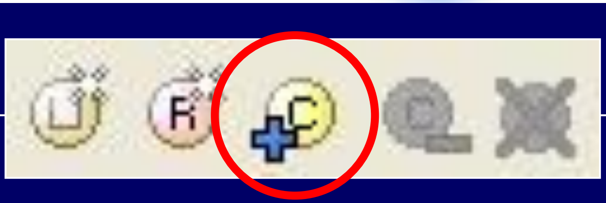
1. *Create a new pizza under NamedPizza*
   either choose from the menu or make it up
2. *Create a new Existential (SomeValuesFrom) Restriction with the hasTopping property and a filler from* **PizzaTopping** *(eg* **HamTopping***)*
3. *Add more Restrictions in the same way to complete the description*
   each restriction is added to an intersection –
   so a Pizza must have toppingA ***and*** *must have toppingB etc*
   see **MargheritaPizza** *for an example*
4. *Create another pizza that has at least one meat ingredient*
   remember disjoints

Polyhierarchies

- By the end of this tutorial we intent to create a **VegetarianPizza**
- Some of our existing Pizzas should be types of **VegetarianPizza**
- However, they could also be types of **SpicyPizza** or **CheeseLoversPizza**
- We need to be able to give them multiple parents

Vegetarian Pizza attempt 1

Start with pizzas2_1.owl



1. *Create a new pizza called "VegetarianPizza" under* **Pizza**
   make this disjoint from its siblings as we have been doing
2. *Select* **MargheritaPizza**
   you will notice that it only has a single parent, **NamedPizza**
3. *Add* **VegetarianPizza** *as a new parent using the conditions widget "Add Named Class" button*
   notice that **MargheritaPizza** *now occurs in 2 places in the asserted hierarchy*
   we have asserted that **MargheritaPizza** *has 2 parents*

Reasoning

- We'd like to be able to check the logical consistency of our model
- We'd also like to make automatic inferences about the subsumption hierarchy. A process known as classifying
  - i.e. Moving classes around in the hierarchy based on their logical definition

- Generic software capable of these tasks are known as reasoners (although you may hear them being referred to as Classifiers)
- RACER, Pellet are reasoners

Running Racer

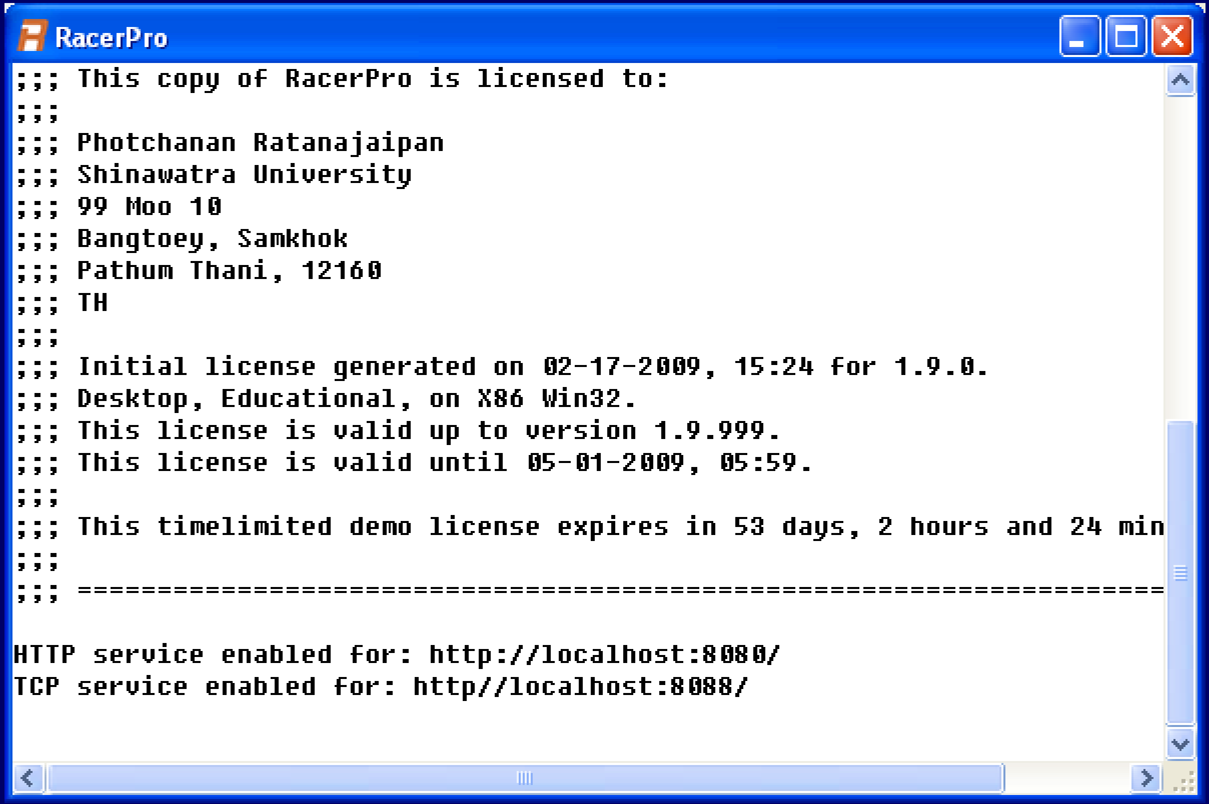
Run racer.exe

A cmd window will open and two “service enabled” messages will appear in the ouput

NB. Alternative DIG reasoners like FaCT, Pellet can also be used

Running Racer

Racer is now ready for use as an http server using a standard interface called DIG

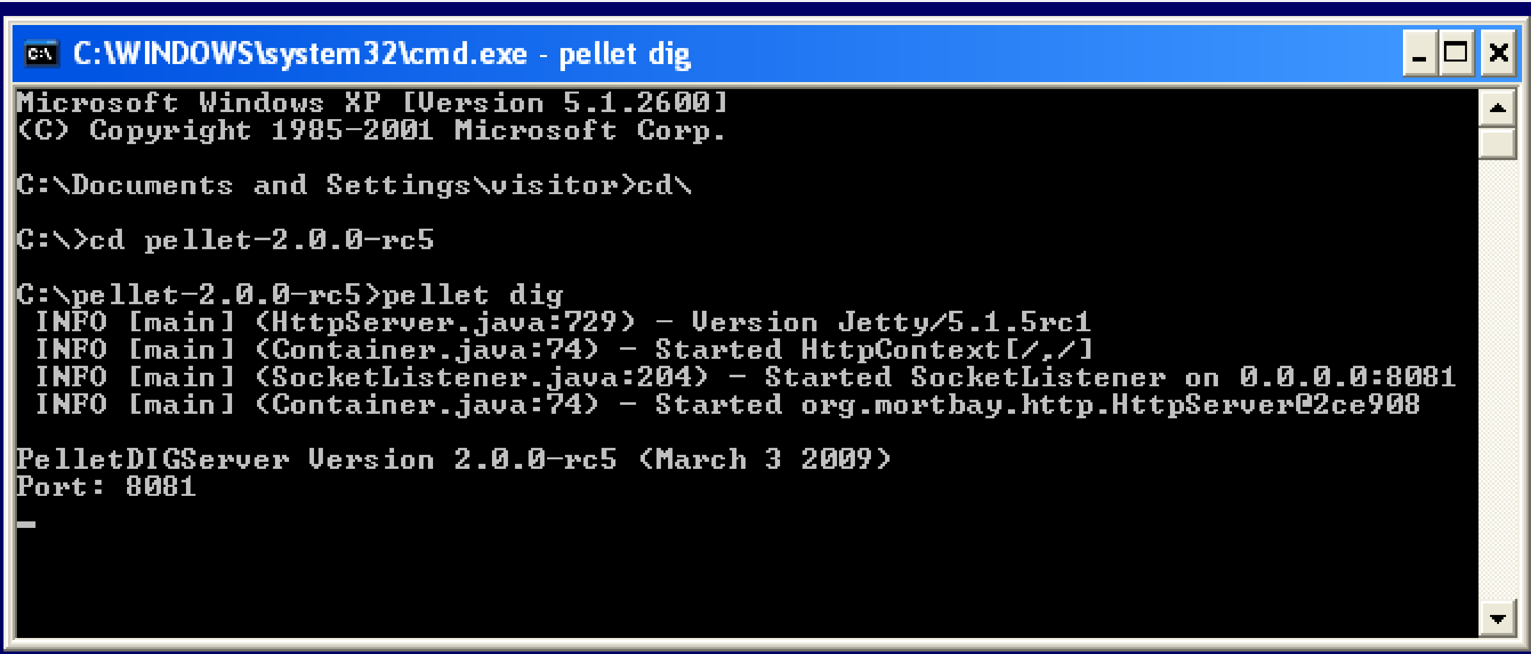


```
;;; This copy of RacerPro is licensed to:
;;;
;;; Photchanan Ratanajaipan
;;; Shinawatra University
;;; 99 Moo 10
;;; Bangtoey, Samkhok
;;; Pathum Thani, 12160
;;; TH
;;;
;;; Initial license generated on 02-17-2009, 15:24 for 1.9.0.
;;; Desktop, Educational, on X86 Win32.
;;; This license is valid up to version 1.9.999.
;;; This license is valid until 05-01-2009, 05:59.
;;;
;;; This timelimited demo license expires in 53 days, 2 hours and 24 min
;;;
;;; ==========================================================================

HTTP service enabled for: http://localhost:8080/
TCP service enabled for: http//localhost:8088/
```

Running Pellet

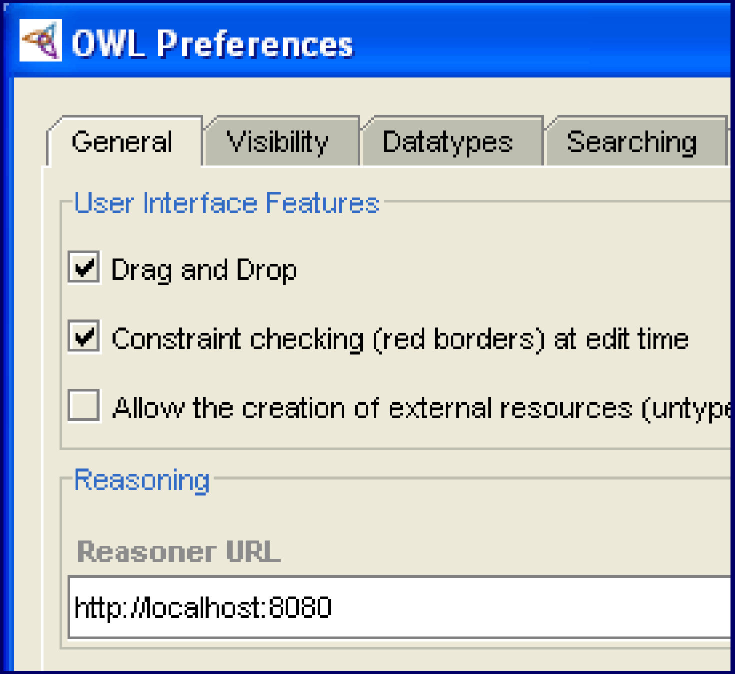
Run “pellet dig”

A cmd window will open, pellet is now ready for use as an http server using a standard interface called DIG



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\visitor>cd\

C:\>cd pellet-2.0.0-rc5

C:\pellet-2.0.0-rc5>pellet dig
 INFO [main] (HttpServer.java:729) - Version Jetty/5.1.5rc1
 INFO [main] (Container.java:74) - Started HttpContext[/,/]
 INFO [main] (SocketListener.java:204) - Started SocketListener on 0.0.0.0:8081
 INFO [main] (Container.java:74) - Started org.mortbay.http.HttpServer@2ce908

PelletDIGServer Version 2.0.0-rc5 (March 3 2009)
Port: 8081
_
```

- You can set the reasoner URL from Preferences setting



OWL Preferences

General | Visibility | Datatypes | Searching

User Interface Features

☑ Drag and Drop

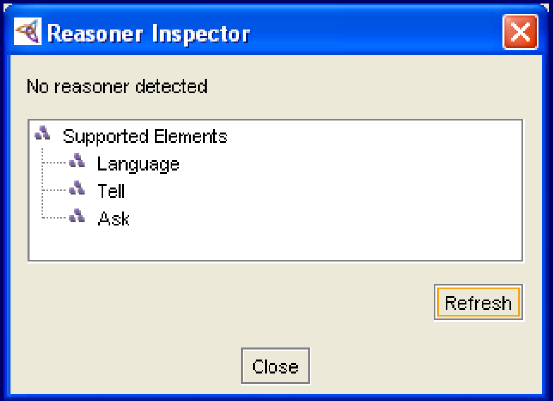
☑ Constraint checking (red borders) at edit time

□ Allow the creation of external resources (untype

Reasoning

Reasoner URL

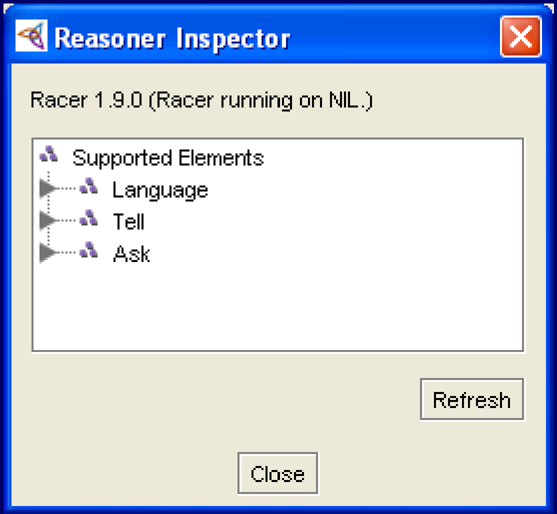
http://localhost:8080



Reasoner Inspector

No reasoner detected

- Supported Elements
  - Language
  - Tell
  - Ask

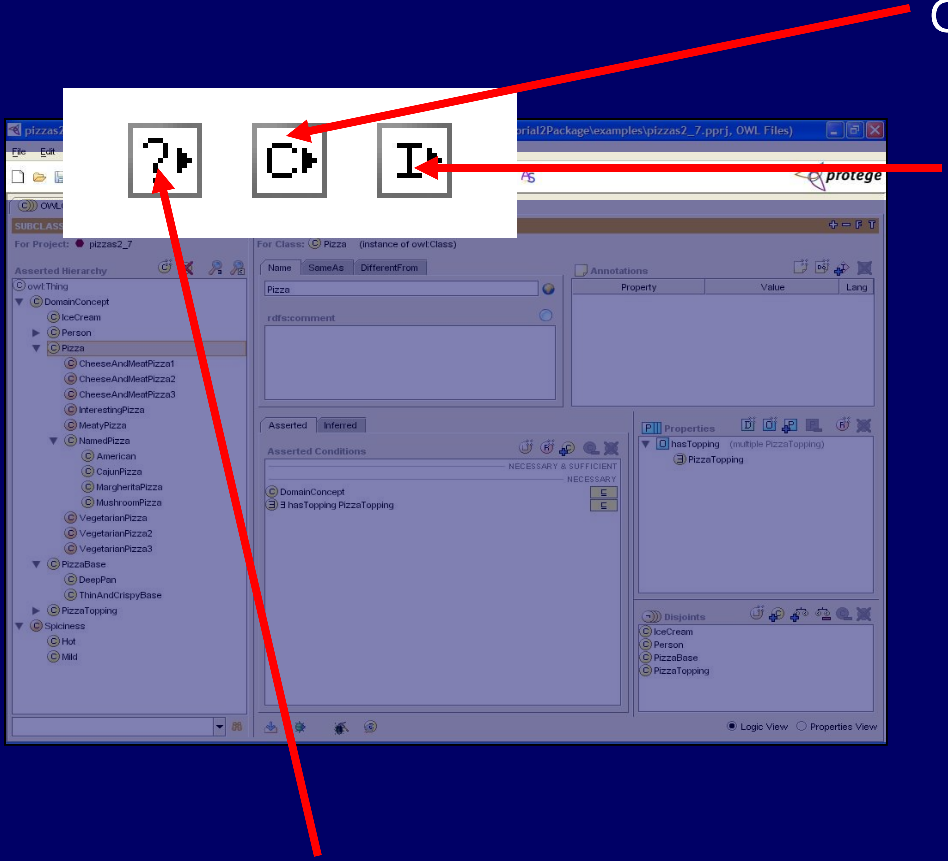


Reasoner Inspector

Racer 1.9.0 (Racer running on NIL.)

- Supported Elements
  - Language
  - Tell
  - Ask

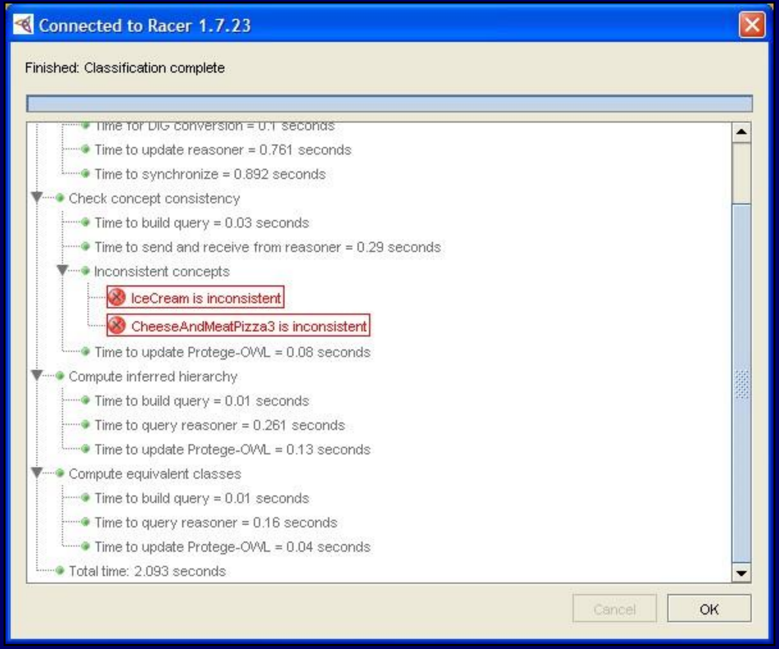
Classifying



Classify taxonomy (and check consistency)

Compute inferred types (for individuals)

Just check consistency (for efficiency)



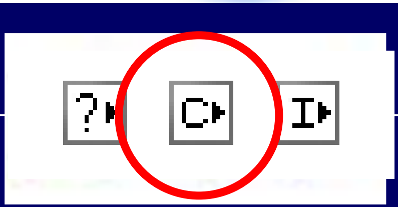
Connected to Racer 1.7.23

Finished: Classification complete

```
    Time for DIG conversion = 0.1 seconds
    Time to update reasoner = 0.761 seconds
    Time to synchronize = 0.892 seconds
Check concept consistency
    Time to build query = 0.03 seconds
    Time to send and receive from reasoner = 0.29 seconds
    Inconsistent concepts
        IceCream is inconsistent
        CheeseAndMeatPizza3 is inconsistent
    Time to update Protege-OWL = 0.08 seconds
Compute inferred hierarchy
    Time to build query = 0.01 seconds
    Time to query reasoner = 0.261 seconds
    Time to update Protege-OWL = 0.13 seconds
Compute equivalent classes
    Time to build query = 0.01 seconds
    Time to query reasoner = 0.16 seconds
    Time to update Protege-OWL = 0.04 seconds
Total time: 2.093 seconds
```

Reasoning about our Pizzas

Start with pizzas2_2.owl

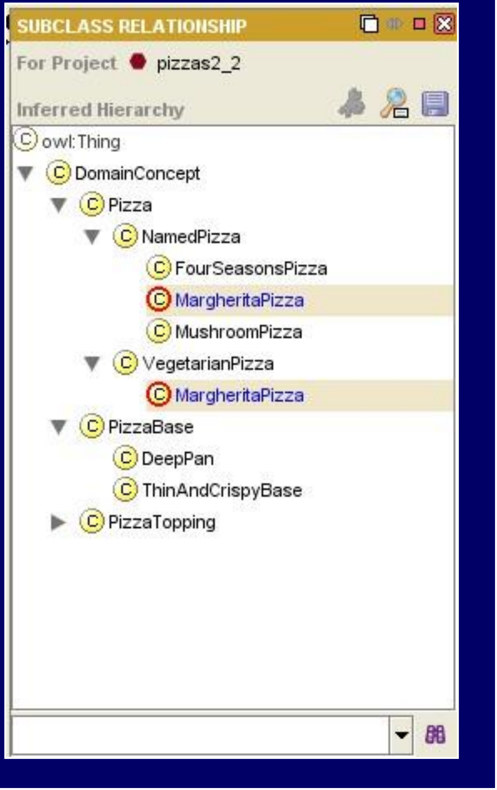


1. *Classify your ontology*

   You will see an inferred hierarchy appear, which will show any movement of classes in the hierarchy

   You will also see a results window appear at the bottom of the screen which describes the results of the reasoner

MargheritaPizza turns out to be inconsistent – why?

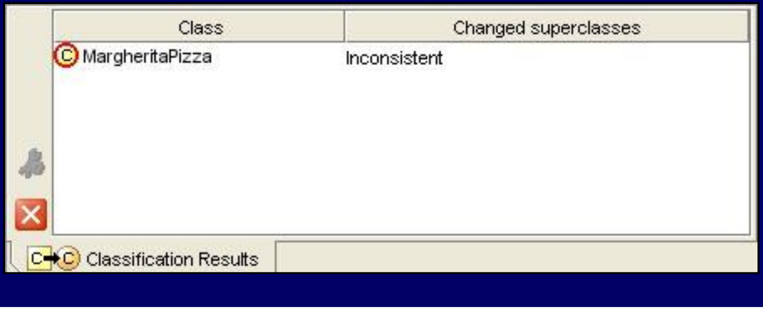


SUBCLASS RELATIONSHIP

For Project ● pizzas2_2

Inferred Hierarchy

```
owl:Thing
▼ DomainConcept
  ▼ Pizza
    ▼ NamedPizza
        FourSeasonsPizza
        MargheritaPizza
        MushroomPizza
    ▼ VegetarianPizza
        MargheritaPizza
  ▼ PizzaBase
      DeepPan
      ThinAndCrispyBase
  ▶ PizzaTopping
```



Class	Changed superclasses
MargheritaPizza	Inconsistent

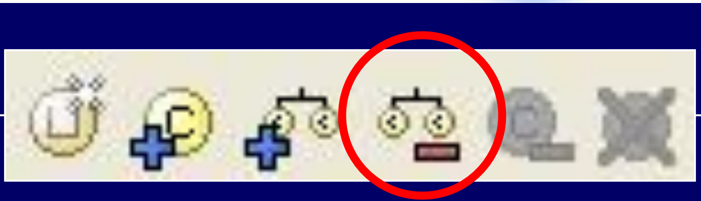
Classification Results

Why is MargheritaPizza inconsistent?

- We are asserting that a **MargheritaPizza** is a subclass of two classes we have stated are disjoint
- The disjoint means nothing can be a **NamedPizza** and a **VegetarianPizza** at the same time
- This means that the class of **MargheritaPizza**s can never contain any individuals
- The class is therefore inconsistent

Attempting again

Start with your current ontology



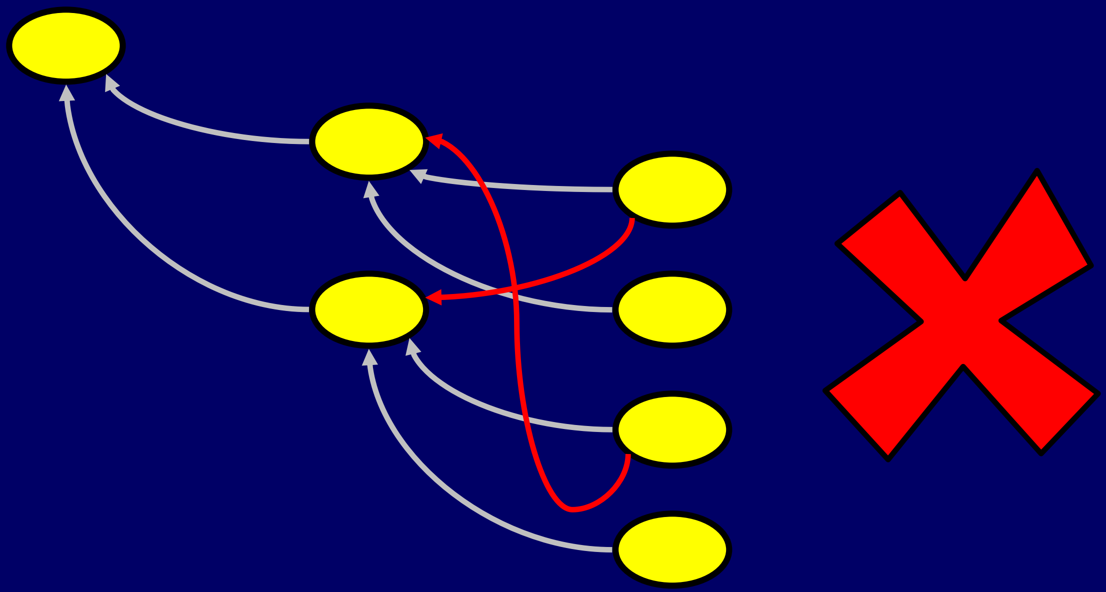
1. *Close the inferred hierarchy and classification results pane*
2. *Remove the disjoint between* **VegetarianPizza** *and its siblings*

   When prompted, choose to remove only between this class and its siblings
3. *Re-Classify your ontology*

   This should now be accepted by the reasoner with no inconsistencies

Asserted Polyhierarchies

- We believe asserting polyhierarchies is bad
- We lose some encapsulation of knowledge
- Difficult to maintain



let the reasoner do it!

Defined Classes

- Have a definition. That is *at least one* Necessary and Sufficient condition
- Are marked in orange in the interface
- Classes, all of whose individuals satisfy this definition, can be inferred to be subclasses
- Reasoners can perform this inference

Describing a MeatyPizza

Start with pizzas2_3.owl, *close the reasoner panes*

1. *Create a subclass of* **Pizza** *called* **MeatyPizza**

   Don’t put in the disjoints or you’ll get the same problems as before

   In general, defined classes are not disjoint

2. *Add a restriction to say:*

   “Every **MeatyPizza** *must have at least one meat topping”*

3. *Classify your ontology*

   What happens?

Defining a MeatyPizza

Start with pizzas2_4.owl, *close the reasoner panes*

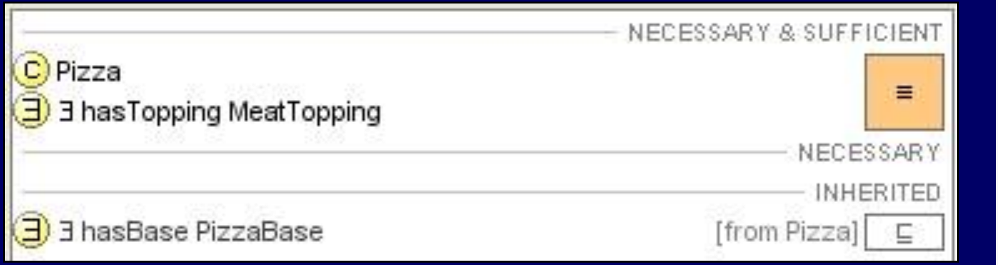
1. *Click and drag your ∃ hasTopping* **MeatTopping** *restriction from* “Necessary” *to* “Necessary & Sufficient”

   The **MeatyPizza** *class now turns orange, denoting that it is now a defined class*

2. *Click and drag the* **Pizza** *Superclass from* “Necessary” *to* “Necessary & Sufficient”

   Make sure when you release you are on top of the existing restriction otherwise you will get 2 sets of conditions.

   You should have a single orange icon on the right stretching across both conditions like this…



NECESSARY & SUFFICIENT

Pizza

∃ hasTopping MeatTopping ≡

NECESSARY

INHERITED

∃ hasBase PizzaBase [from Pizza] ⊑

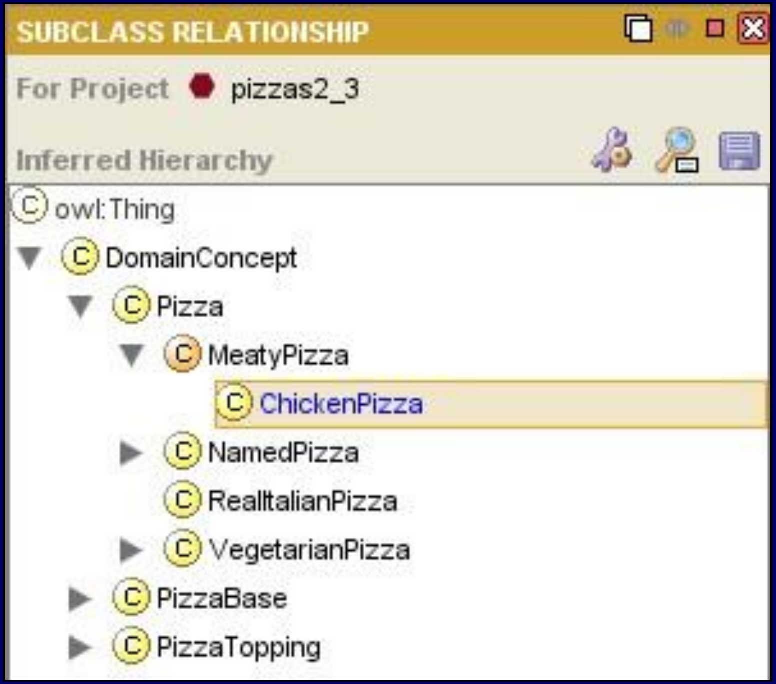
3. *Classify your ontology*

   What happens?

Reasoner Classification

- The reasoner has been able to infer that anything that is a **Pizza** that has at least one topping from **MeatTopping** is a **MeatyPizza**

- Therefore, classes fitting this definition are found to be subclasses of **MeatyPizza**, or are subsumed by **MeatyPizza**
- The inferred hierarchy is updated to reflect this and moved classes are highlighted in blue



SUBCLASS RELATIONSHIP

For Project ● pizzas2_3

Inferred Hierarchy

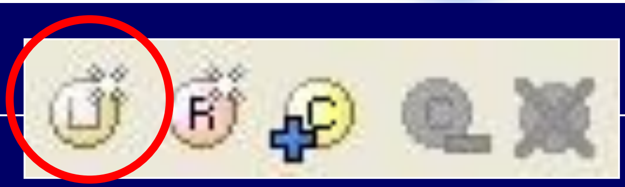
```
(C) owl:Thing
▼ (C) DomainConcept
    ▼ (C) Pizza
        ▼ (C) MeatyPizza
            (C) ChickenPizza
        ▶ (C) NamedPizza
          (C) RealItalianPizza
        ▶ (C) VegetarianPizza
    ▶ (C) PizzaBase
    ▶ (C) PizzaTopping
```

How do we Define a Vegetarian Pizza?

- Nasty
- Define in words?
  - “a pizza with only vegetarian toppings”?
  - “a pizza with no meat (or fish) toppings”?
  - “a pizza that is not a MeatyPizza”?
- More than one way to model this

Defining a Vegetarian Topping

Start with pizzas2_5.owl



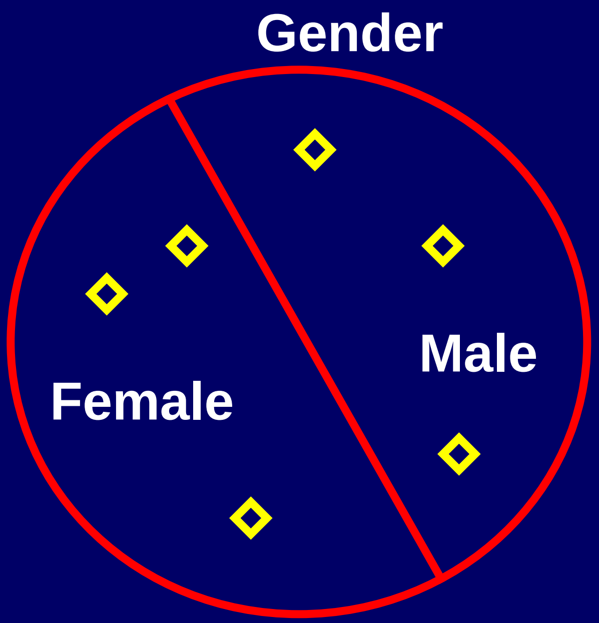
1. *Create a subclass of* **PizzaTopping** *called* **VegetarianTopping**
2. *Click "Create New Expression" in the Conditions Widget*
   Type in or select each of the top level **PizzaToppings** *that are not meat or fish (ie* **DairyTopping**, **FruitTopping** *etc) and between each, type the word "or"*
   the "or" will be translated into a union symbol
3. *Press Return when finished*
   you have created an anonymous class described by the expression
4. *Make this a defined class by moving both conditions from the "Necessary" to the "Necessary & Sufficient" conditions*
5. *Classify your ontology*

Class Constructors: Union

- AKA “disjunction”
- This OR That OR TheOther
- (This $\sqcup$ That $\sqcup$ TheOther)
- Set theory
- Commonly used for:
  - – Covering axioms (like **VegetarianTopping**)
  - – Closure

Covering Axioms

- Covered class – that to which the condition is added
- Covering classes – those in the union expression
- A covering axiom in the "Necessary & Sufficient" Conditions means: the covered class cannot contain any instances from a class other than one of the covering classes



$\text{Gender} \equiv \text{Female} \sqcup \text{Male}$

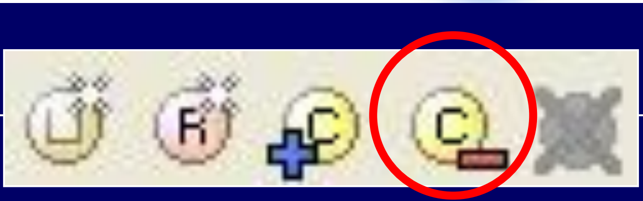
In this example, the class Gender is "covered" by Male or Female

All individuals in Gender must be individuals from Male or Female

There are no other types of Gender

Vegetarian Pizza attempt 2

Start with pizzas2_6.owl



1. *Select* **MargheritaPizza** *and remove* **VegetarianPizza** *from its superclasses*
2. *Select* **VegetarianPizza** *and create a restriction to say that it "only has toppings from* **VegetarianTopping***"*
3. *Make this a defined class by moving all conditions from* "Necessary" *to* "Necessary & Sufficient"

   Make sure when you release you are on top of the existing restriction otherwise you will get 2 sets of conditions.

   You should have a single orange icon on the right stretching across both conditions
4. *Classify your ontology*

   What happens?

Open World Assumption

- The reasoner does not have enough information to classify pizzas under **VegetarianPizza**
- Typically several Existential restrictions on a single property with different fillers – like primitive pizzas
- Existential should be paraphrased by “amongst other things…”
- Must state that a description is complete
- We need closure for the given property
- This is in the form of a Universal Restriction with a Union of the other fillers using that property

Closure

- Example: **MargheritaPizza**

  All **MargheritaPizzas** must have:

  - at least 1 topping from **MozzarellaTopping** and
  - at least 1 topping from **TomatoTopping** and
  - only toppings from **MozzarellaTopping** or **TomatoTopping**

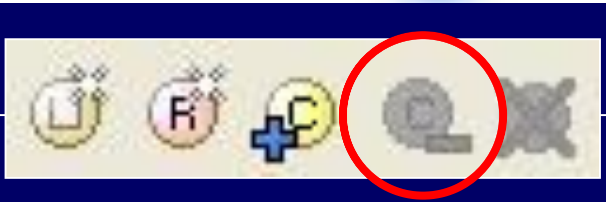
- The last part is paraphrased into

  “no other toppings”

- The union closes the hasTopping property on **MargheritaPizza**

Closing Pizza Descriptions

Start with pizzas2_7.owl



1. *Select* **MargheritaPizza**
2. *Create a Universal Restriction on the hasTopping property with a filler of "**TomatoTopping ⊔ MozzarellaTopping**"*

   Remember, you can type "or" to achieve this, or you can use the expression palette
3. *Close your other pizzas*

   Each time you need to create a filler with the union of all the classes used on the hasTopping property (ie all the toppings used on that pizza)
4. *Classify your ontology*

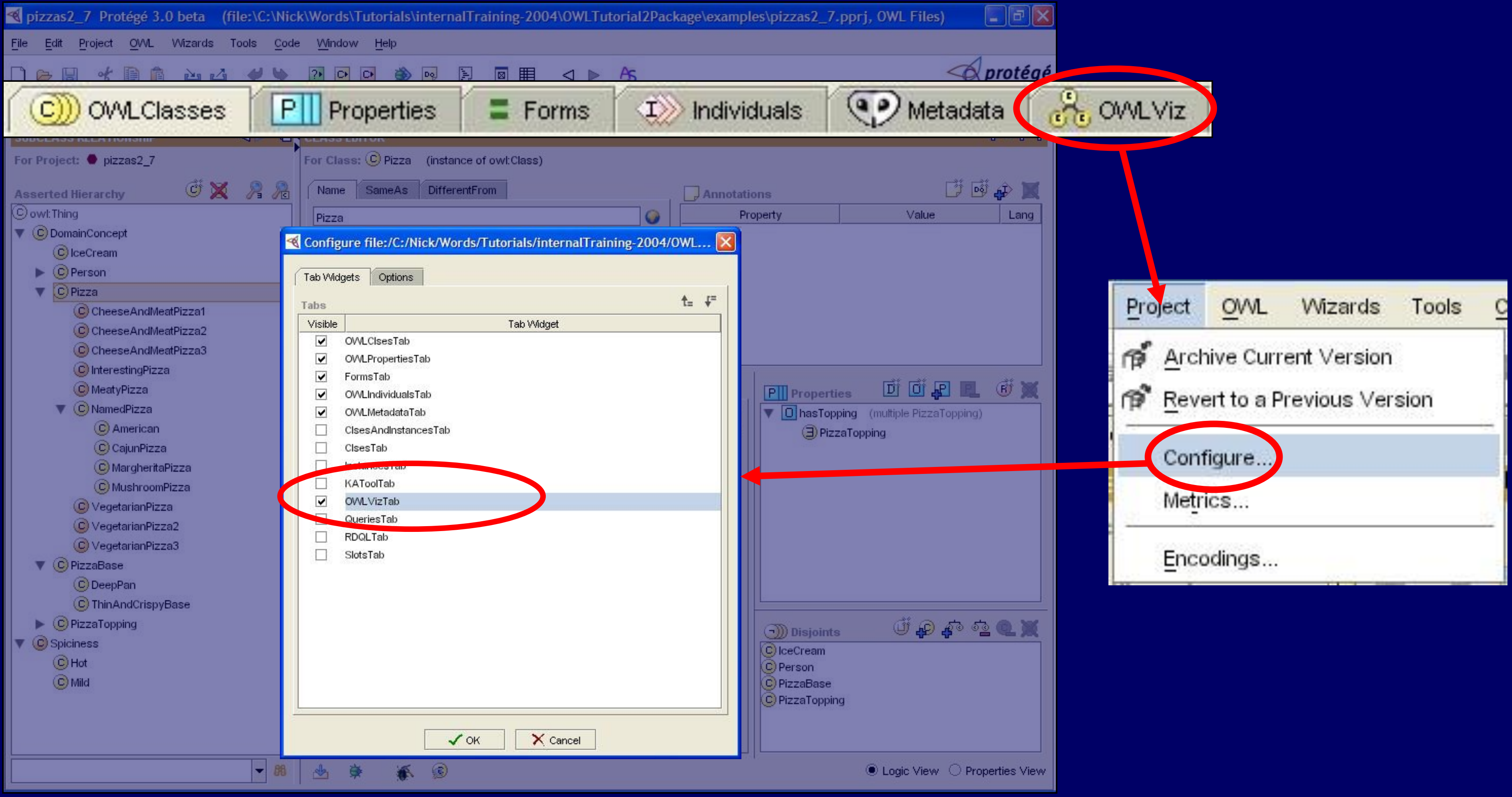
   Finally, the defined class **VegetarianPizza** *should subsume any classes that only have vegetarian toppings*

Summary

You should now be able to:

- Use Defined Classes allow a polyhierarchy to be computed
- Classify and check consistency using a Reasoner
- Create Covering Axioms
- Close Class Descriptions to cope with Open World Reasoning

Viewing our Hierarchy Graphically



Configure file:/C:/Nick/Words/Tutorials/internalTraining-2004/OWL...

Tab Widgets | Options

Tabs

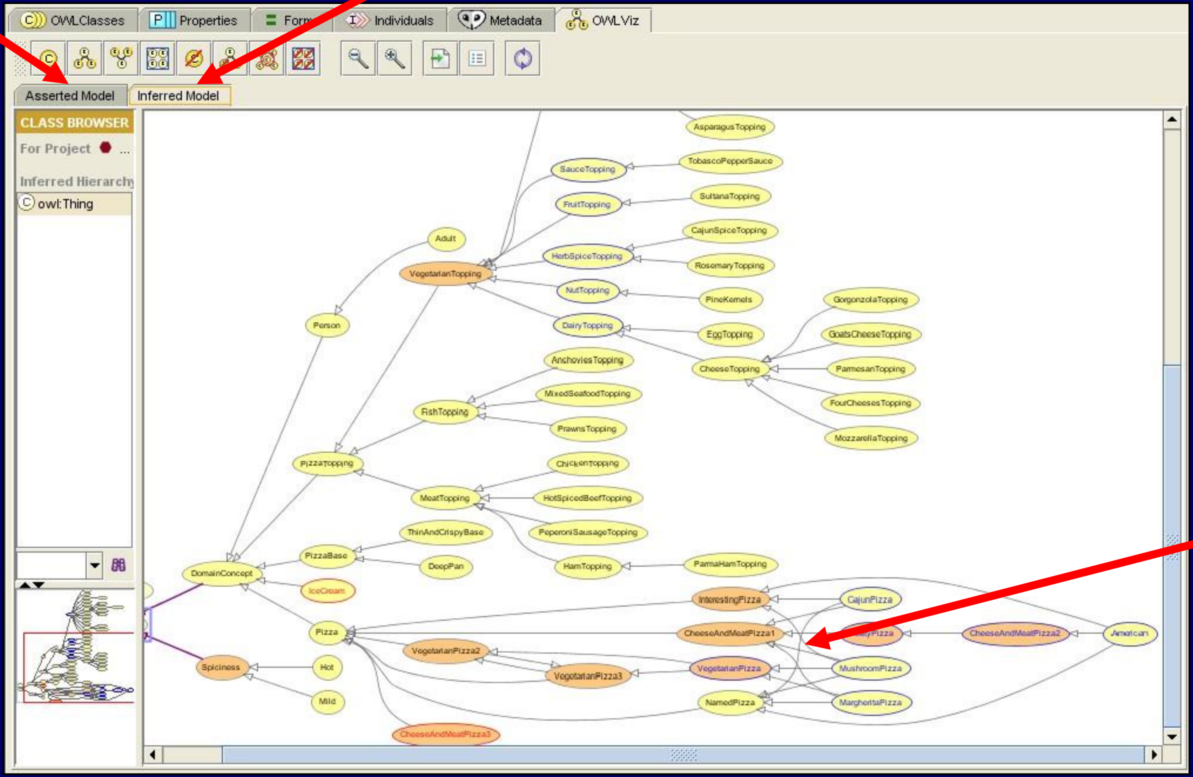
Visible	Tab Widget
☑	OWLClsesTab
☑	OWLPropertiesTab
☑	FormsTab
☑	OWLIndividualsTab
☑	OWLMetadataTab
□	ClsesAndInstancesTab
□	ClsesTab
□	InstancesTab
□	KAToolTab
☑	OWLVizTab
□	QueriesTab
□	RDQLTab
□	SlotsTab

OK Cancel

OWLViz Tab

View Asserted Model

View Inferred Model



Polyhierarchy tangle

Your Pizza Finder

- Once you have a pizza ontology you are happy with, you can “plug it in” to the PizzaFinder
- Instructions available on line at…

Other Exercises: Create a ProteinLoversPizza

Start with pizzas2_8.owl

- *Create a new subclass of* **Pizza**
- *Define this as:*
  "Any **Pizza** *that has at least one* **MeatTopping** *and at least one* **CheeseTopping** *and at least one* **FishTopping***"*
- *If you don't have any pizzas that will classify under this, create one which should (***SicilianaPizza** *should)*
- *Classify to check that it works*

Other Exercises: Define RealItalianPizza

Start with pizzas2_9.owl

- *Convert* **RealItalianPizza** *to a defined class*
- *Add information to your pizzas to allow some of them to classify under this one*
- *Classify*

  remember to check your disjoint if you have problems

Others

- Show RDF/XML source code
- OWLViz Tab
- Protégé OWL Reasoner API http://protege.stanford.edu/plugins/owl/api/ReasonerAPIExamples.html
- Ontology Development
- GiftMe – The Gift Recommendation System

Thank You

- Feedback on tutorial appreciated

- Original of PowerPoint slides available from
  - http://www.cs.man.ac.uk/~drummond/cs646

- Software / resources / community at:
  - http://www.co-ode.org/
  - http://protege.stanford.edu/