



# Chapter 1

## Introduction

This guide introduces Protégé 4 for creating OWL ontologies. Chapter 3 gives a brief overview of the OWL ontology language. Chapter 4 focuses on building an OWL-DL ontology and using a Description Logic Reasoner to check the consistency of the ontology and automatically compute the ontology class hierarchy. Chapter 7 describes some OWL constructs such as hasValue Restrictions and Enumerated classes, which aren't directly used in the main tutorial.

### 1.1 Conventions

Class, property and individual names are written in a sans serif font **like this**.


Names for user interface views are presented in a style '**like this**'.

Where exercises require information to be typed into Protégé 4 a type writer font is used **like this**.

Exercises and required tutorial steps are presented like this:

---

**Exercise 1: Accomplish this**

- 
1. Do this.
  2. Then do this.
  3. Then do this.
-



Tips and suggestions related to using Protégé 4 and building ontologies are presented like this.

#### MEANING



Explanation as to what things mean are presented like this.



Potential pitfalls and warnings are presented like this.



#### NOTE

General notes are presented like this.

#### Vocabulary



Vocabulary explanations and alternative names are presented like this.

## Chapter 2

# Requirements

In order to follow this tutorial you must have Protégé 4, which is available from the Protégé website <sup>1</sup>, and the Protégé Plugins which are available via the CO-ODE web site <sup>2</sup>. It is also recommended (but not necessary) to use the OWLViz plugin, which allows the asserted and inferred classification hierarchies to be visualised, and is available from the CO-ODE web site, or can be installed when Protégé 4 is installed. For installation steps, please see the documentation for each component.

---

<sup>1</sup><http://protege.stanford.edu>

<sup>2</sup><http://www.co-ode.org>

## Chapter 3

# What are OWL Ontologies?

Ontologies are used to capture knowledge about some domain of interest. An ontology describes the concepts in the domain and also the relationships that hold between those concepts. Different ontology languages provide different facilities. The most recent development in standard ontology languages is OWL from the World Wide Web Consortium (W3C)<sup>1</sup>. Like Protégé, OWL makes it possible to describe concepts but it also provides new facilities. It has a richer set of operators - e.g. intersection, union and negation. It is based on a different logical model which makes it possible for concepts to be defined as well as described. Complex concepts can therefore be built up in definitions out of simpler concepts. Furthermore, the logical model allows the use of a reasoner which can check whether or not all of the statements and definitions in the ontology are mutually consistent and can also recognise which concepts fit under which definitions. The reasoner can therefore help to maintain the hierarchy correctly. This is particularly useful when dealing with cases where classes can have more than one parent.

### 3.1 Components of OWL Ontologies

OWL ontologies have similar components to Protégé frame based ontologies. However, the terminology used to describe these components is slightly different from that used in Protégé. An OWL ontology consists of Individuals, Properties, and Classes, which roughly correspond to Protégé frames Instances, Slots and Classes.

#### 3.1.1 Individuals

Individuals, represent objects in the domain in which we are interested<sup>2</sup>. An important difference between Protégé and OWL is that OWL does not use the Unique Name Assumption (UNA). This means that two different names could actually refer to the same individual. For example, “Queen Elizabeth”, “The Queen” and “Elizabeth Windsor” *might* all refer to the same individual. In OWL, it must be explicitly stated that individuals are the same as each other, or different to each other — otherwise they *might* be the same as each other, or they *might* be different to each other. Figure 3.1 shows a representation of some individuals in some domain—in this tutorial we represent individuals as diamonds in diagrams.

---

<sup>1</sup><http://www.w3.org/TR/owl-guide/>

<sup>2</sup>Also known as *the domain of discourse*.

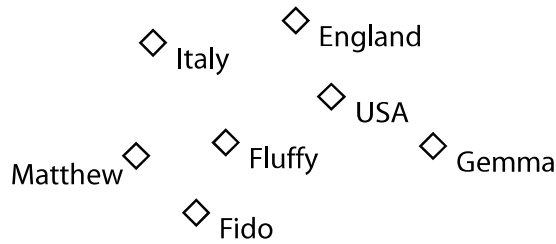


Figure 3.1: Representation Of Individuals

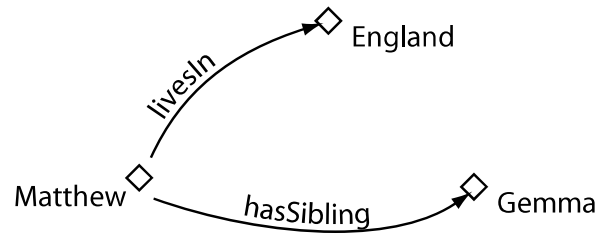


Figure 3.2: Representation Of Properties



Individuals are also known as *instances*. Individuals can be referred to as being ‘instances of classes’.

### 3.1.2 Properties

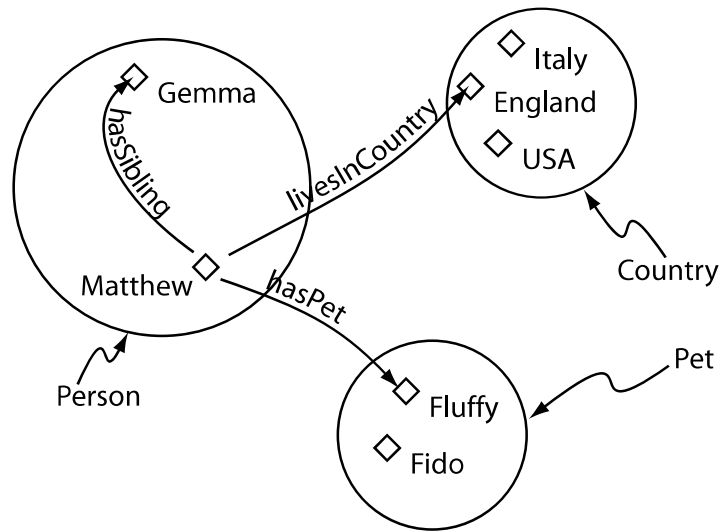
Properties are *binary* relations<sup>3</sup> on *individuals* - i.e. properties link *two* individuals together<sup>4</sup>. For example, the property **hasSibling** might link the individual **Matthew** to the individual **Gemma**, or the property **hasChild** might link the individual **Peter** to the individual **Matthew**. Properties can have inverses. For example, the inverse of **hasOwner** is **isOwnedBy**. Properties can be limited to having a single value – i.e. to being *functional*. They can also be either *transitive* or *symmetric*. These ‘property characteristics’ are explained in detail in Section 4.8. Figure 3.2 shows a representation of some properties linking some individuals together.



Properties are roughly equivalent to *slots* in Protégé. They are also known as *roles* in description logics and *relations* in UML and other object oriented notions. In GRAIL and some other formalisms they are called *attributes*.

<sup>3</sup>A binary relation is a relation between *two* things.

<sup>4</sup>Strictly speaking we should speak of ‘instances of properties’ linking individuals, but for the sake of brevity we will keep it simple.



**Figure 3.3:** Representation Of Classes (Containing Individuals)

### 3.1.3 Classes

OWL classes are interpreted as *sets* that contain individuals. They are *described* using formal (mathematical) descriptions that state precisely the requirements for membership of the class. For example, the class **Cat** would contain all the individuals that are cats in our domain of interest.<sup>5</sup> Classes may be organised into a superclass-subclass hierarchy, which is also known as a *taxonomy*. Subclasses specialise ('are subsumed by') their superclasses. For example consider the classes **Animal** and **Cat** – **Cat** might be a subclass of **Animal** (so **Animal** is the superclass of **Cat**). This says that, 'All cats are animals', 'All members of the class **Cat** are members of the class **Animal**', 'Being a **Cat** implies that you're an **Animal**', and '**Cat** is *subsumed* by **Animal**'. One of the key features of OWL-DL is that these superclass-subclass relationships (subsumption relationships) can be computed automatically by a *reasoner* – more on this later. Figure 3.3 shows a representation of some classes containing individuals – classes are represented as circles or ovals, rather like sets in Venn diagrams.



The word *concept* is sometimes used in place of class. Classes are a concrete representation of concepts.

In OWL classes are built up of descriptions that specify the conditions that must be satisfied by an individual for it to be a member of the class. How to formulate these descriptions will be explained as the tutorial progresses.

<sup>5</sup>Individuals may belong to more than one class.

## Chapter 4

# Building An OWL Ontology

This chapter describes how to create an ontology of Pizzas. We use Pizzas because we have found them to provide many useful examples.<sup>1</sup>

---

### Exercise 2: Create a new OWL Ontology

---

1. Start Protégé
  2. When the Welcome To Protégé dialog box appears, press the **‘Create New OWL Ontology’**.
  3. A **‘Create Ontology URI Wizard** will appear’. Every ontology is named using a Unique Resource Identifier (URI). Replace the default URI with `http://www.pizza.com/ontologies/pizza.owl` and press **‘Next’**.
  4. You will also want to save your Ontology to a file on your PC. You can browse your hard disk and save your ontology to a new file, you might want to name your file **‘pizza.owl’**. Once you choose a file press **‘Finish’**.
- 

After a short amount of time, a new empty Protégé file will have been created and the **‘Active Ontology Tab’** shown in Figure 4.1 will be visible. As can be seen from Figure 4.1, the **‘Active Ontology Tab’** allows information about the ontology to be specified. For example, the ontology URI can be changed, annotations on the ontology such as comments may be added and edited, and namespaces and imports can be set up via this tab.

---

<sup>1</sup>The Ontology that we will create is based upon a Pizza Ontology that has been used as the basis for a course on editing DAML+OIL ontologies in OilEd (<http://oiled.man.ac.uk>), which was taught at the University Of Manchester.



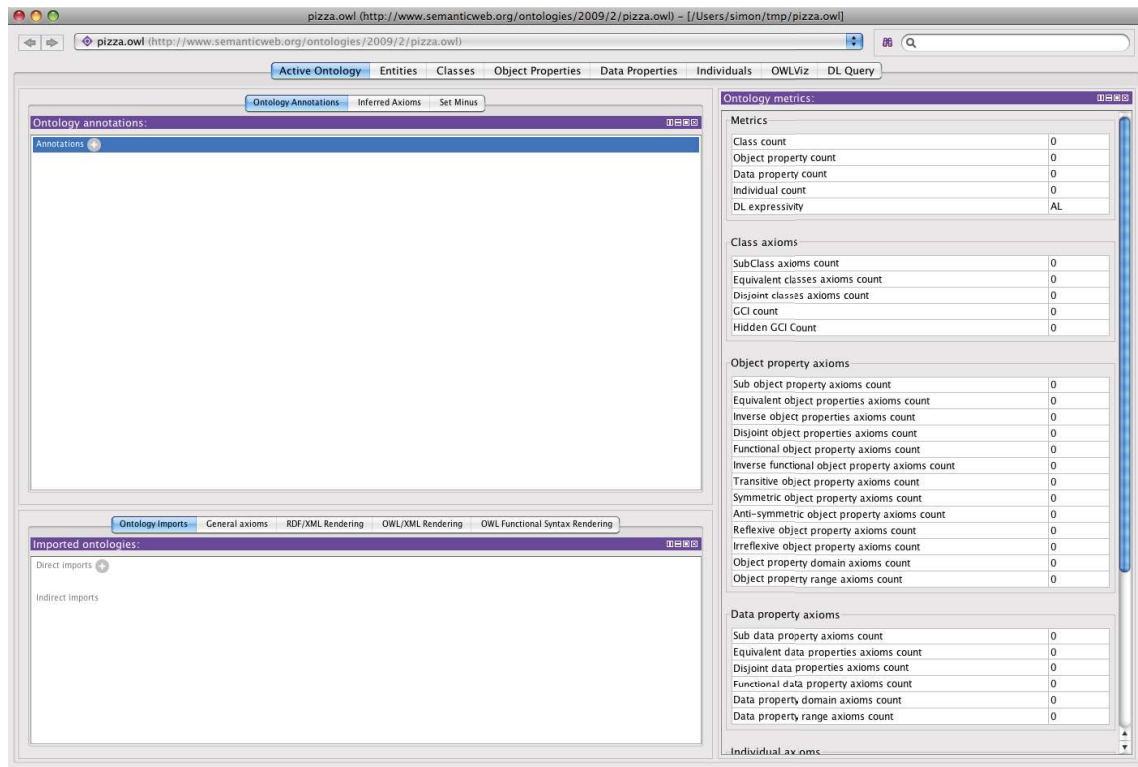
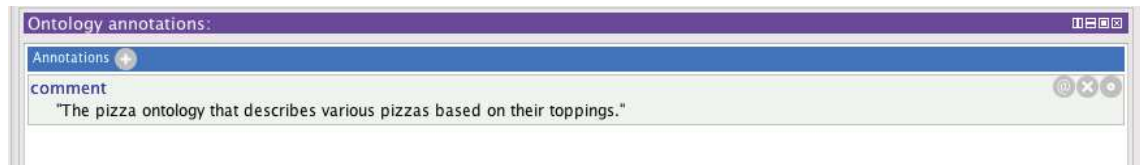


Figure 4.1: The Active Ontology Tab



**Figure 4.2:** The Ontology Annotations View – The ontology has a comment as indicated by the `comment` annotation

### Exercise 3: Add a comment to the ontology

---

1. Ensure that the ‘**Active Ontology Tab**’ is selected.
  2. In the ‘**Ontology Annotations**’ view, click the ‘**Add**’ icon (+) next to Annotations. An editing window will appear in the table. Select ‘comment’ from the list of built in annotation URIs and type your comment in the text box in the right hand pane.
  3. Enter a comment such as **A pizza ontology that describes various pizzas based on their toppings.** and press OK to assign the comment. The annotations view on the ‘**Active Ontology Tab**’ should look like the picture shown in Figure 4.2
- 

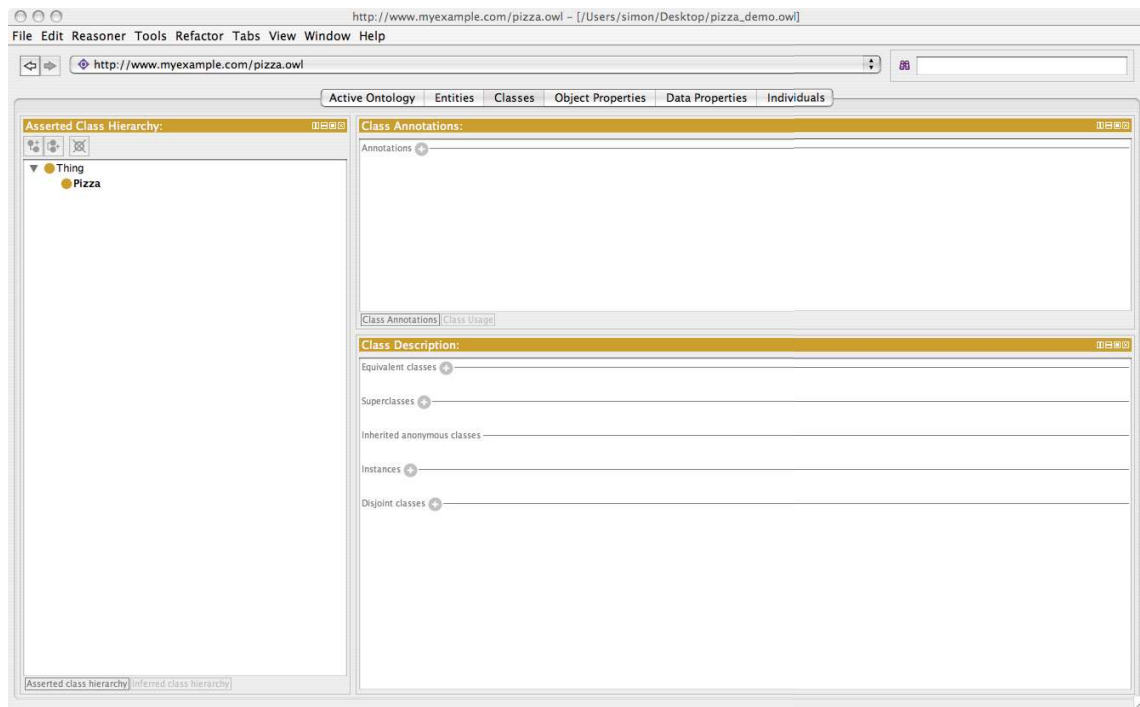
## 4.1 Named Classes

As mentioned previously, an ontology contains classes – indeed, the main building blocks of an OWL ontology are classes. In Protégé 4, editing of classes is carried out using the ‘**Classes Tab**’ shown in Figure 4.3. The initial class hierarchy tree view should resemble the picture shown in Figure 4.4. The empty ontology contains one class called **Thing**. As mentioned previously, OWL classes are interpreted as sets of *individuals* (or sets of objects). The class **Thing** is the class that represents the set containing *all* individuals. Because of this all classes are subclasses of **Thing**.<sup>2</sup>

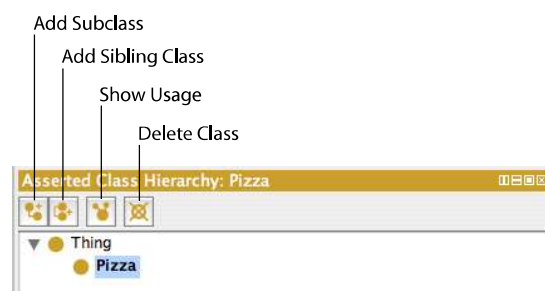
Let’s add some classes to the ontology in order to define what we believe a pizza to be.

---

<sup>2</sup>Thing is part of the OWL Vocabulary, which is defined by the ontology located at <http://www.w3.org/2002/07/owl/#>



**Figure 4.3:** The Classes Tab



**Figure 4.4:** The Class Hierarchy Pane

#### Exercise 4: Create classes Pizza, PizzaTopping and PizzaBase

---

1. Ensure that the ‘**Classes Tab**’ is selected.
  2. Press the ‘**Add**’ icon (⊕) shown in Figure 4.4. This button creates a new class as a subclass of the selected class (in this case we want to create a subclass of **Thing**).
  3. A dialog will appear for you to name your class, enter **Pizza** (as shown in Figure 4.5) and hit return.
  4. Repeat the previous steps to add the classes **PizzaTopping** and also **PizzaBase**, ensuring that **Thing** is selected before the ‘**Add**’ icon (⊕) is pressed so that the classes are created as subclasses of **Thing**.
- 

The class hierarchy should now resemble the hierarchy shown in Figure 4.6.



After creating **Pizza**, instead of re-selecting **Thing** and using the ‘**Create subclass**’ button to create **PizzaTopping** and **PizzaBase** as further subclasses of **Thing**, the ‘**Add sibling class**’ button (shown in Figure 4.4) can be used. While **Pizza** is selected, use the ‘**Create sibling class**’ button to create **PizzaTopping** and then use this button again (while **PizzaTopping** is selected) to create **PizzaBase** as sibling classes of **PizzaTopping** – these classes will of course still be created as subclasses of **Thing**, since **Pizza** is a subclass of **Thing**.

Vocabulary



A class hierarchy may also be called a taxonomy.



Although there are no mandatory naming conventions for OWL classes, we recommend that all class names should start with a capital letter and should not contain spaces. (This kind of notation is known as CamelBack notation and is the notation used in this tutorial). For example **Pizza**, **PizzaTopping**, **MargheritaPizza**. Alternatively, you can use underscores to join words. For example **Pizza\_Topping**. Which ever convention you use, it is important to be consistent.

## 4.2 Disjoint Classes

Having added the classes **Pizza**, **PizzaTopping** and **PizzaBase** to the ontology, we now need to say these classes are *disjoint*, so that an individual (or object) cannot be an instance of more than one of these



Figure 4.5: Class Name Dialog

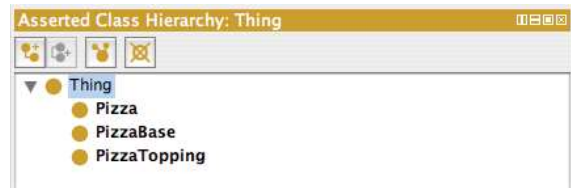


Figure 4.6: The Initial Class Hierarchy

three classes. To specify classes that are disjoint from the selected class click the ‘**Disjoints classes**’ button which is located at the bottom of the ‘**Class Description**’ view.

#### Exercise 5: Make Pizza, PizzaTopping and PizzaBase disjoint from each other

1. Select the class **Pizza** in the class hierarchy.
2. Press the ‘**Disjoint classes**’ button in the ‘**class description**’ view, this will bring up a dialog where you can select multiple classes to be disjoint. This will make **PizzaBase** and **PizzaTopping** (the sibling classes of **Pizza**) disjoint from **Pizza**.

Notice that the disjoint classes view now displays **PizzaTopping** and **PizzaBase**. Select the class **PizzaBase**. Notice that the disjoint classes view displays the classes that are now disjoint to **PizzaBase**, namely **Pizza** and **PizzaTopping**.

#### MEANING



OWL Classes are assumed to ‘overlap’. We therefore cannot assume that an individual is not a member of a particular class simply because it has not been *asserted* to be a member of that class. In order to ‘separate’ a group of classes we must make them disjoint from one another. This ensures that an individual which has been asserted to be a member of one of the classes in the group cannot be a member of any other classes in that group. In our above example **Pizza**, **PizzaTopping** and **PizzaBase** have been made disjoint from one another. This means that it is not possible for an individual to be a member of a combination of these classes – it would not make sense for an individual to be a **Pizza** and a **PizzaBase**!

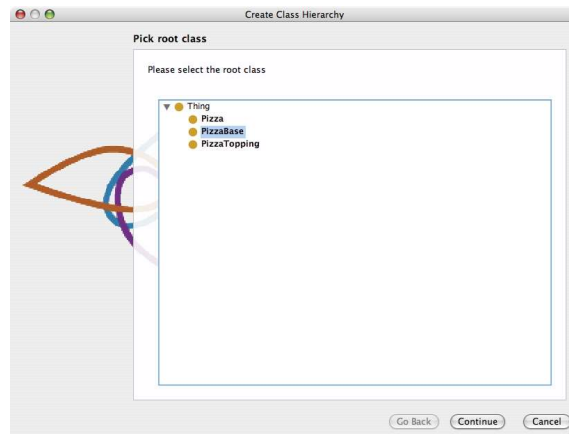


Figure 4.7: Create Class Hierarchy: Select class page

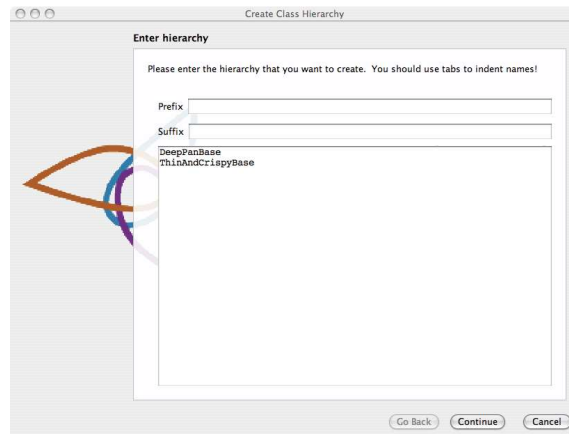


Figure 4.8: Create Class Hierarchy: Enter classes page

### 4.3 Using Create Class Hierarchy To Create Classes

In this section we will use the 'Create Class Hierarchy' tool to add some subclasses of the class `PizzaBase`.

### Exercise 6: Use the ‘Create Class Hierarchy’ Tool to create ThinAndCrispy and DeepPan as subclasses of PizzaBase

---

1. Select the class **PizzaBase** in the class hierarchy.
  2. From the Tools menu on the Protégé menu bar select ‘**Create Class Hierarchy...**’.
  3. The tools shown in Figure 4.7 will appear. Since we preselected the **PizzaBase** class, the first radio button at the top of the tool should be prompting us to create the classes under the class **PizzaBase**. If we had not preselected **PizzaBase** before starting the tool, then the tree could be used to select the class.
  4. Press the ‘**Next**’ button on the tool—The page shown in Figure 4.8 will be displayed. We now need to tell the tool the subclasses of **PizzaBase** that we want to create. In the large text area, type in the class name **ThinAndCrispyBase** (for a thin based pizza) and hit return. Also enter the class name **DeepPanBase** so that the page resembles that shown in Figure 4.8 .
  5. Hit the ‘**Next**’ button on the tool. The tool checks that the names entered adhere to the naming styles that have previously been mentioned (No spaces etc.). It also checks for uniqueness – no two class names may be the same. If there are any errors in the class names, they will be presented on this page, along with suggestions for corrections.
  6. Hit the ‘**Next**’ button on the tool. Ensure the tick box ‘**Make all new classes disjoint**’ is *ticked* — instead of having to use the disjoint classes view, the tool will automatically make the new classes disjoint for us.
- 

After the ‘**Next**’ button has been pressed, the tool creates the classes, makes them disjoint. Click ‘**Finish**’ to dismiss the tool. The ontology should now have **ThinAndCrispyBase** and also **DeepPanBase** as subclasses of **PizzaBase**. These new classes should be disjoint to each other. Hence, a pizza base cannot be both thin and crispy *and* deep pan. It isn’t difficult to see that if we had a lot of classes to add to the ontology, the tool would dramatically speed up the process of adding them.



On page one of the ‘**Create class hierarchy wizard**’ the classes to be created are entered. If we had a lot of classes to create that had the same prefix or suffix we could use the options to auto prepend and auto append text to the class names that we entered.

## Creating Some Pizza Toppings

Now that we have some basic classes, let’s create some pizza toppings. In order to be useful later on the toppings will be grouped into various categories — meat toppings, vegetable toppings, cheese toppings

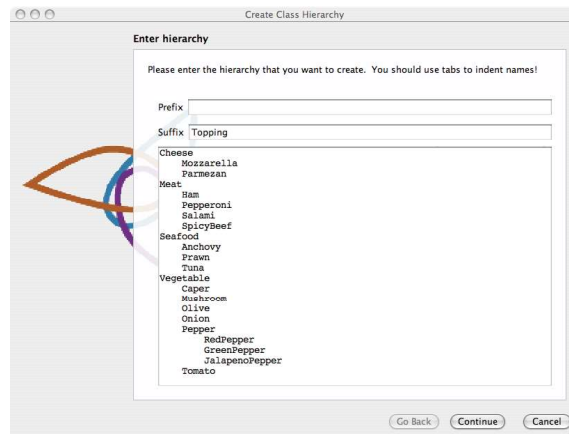


Figure 4.9: Topping Hierarchy

and seafood toppings.

### Exercise 7: Create some subclasses of PizzaTopping

1. Select the class **PizzaTopping** in the class hierarchy.
2. Invoke the ‘**Create class hierarchy...**’ tool in the same way as the tool was started in the previous exercise.
3. Ensure **PizzaTopping** is selected and press the ‘**Next**’ button.
4. We want all our topping classes to end in **topping**, so in the ‘**Suffix all in list with**’ field, enter **Topping**. The tool will save us some typing by automatically appending **Topping** to all of our class names.
5. The tool allows a hierarchy of classes to be entered using a tab indented tree. Using the text area in the tool, enter the class names as shown in Figure 4.9. Note that class names must be indented using tabs, so for example **SpicyBeef**, which we want to be a subclass of **Meat** is entered under **Meat** and indented with a tab. Likewise, **Pepperoni** is also entered under **Meat** below **SpicyBeef** and also indented with a tab.
6. Having entered a tab indented list of classes, press the ‘**Next**’ button and then make sure that ‘**Make all primitive siblings disjoint**’ check box is ticked so that new *sibling* classes are made disjoint with each other.
7. Press the ‘**Finish**’ button to create the classes. Press ‘**Finish**’ again to close the tool.

The class hierarchy should now look similar to that shown in Figure 4.10 (the ordering of classes may be slightly different).



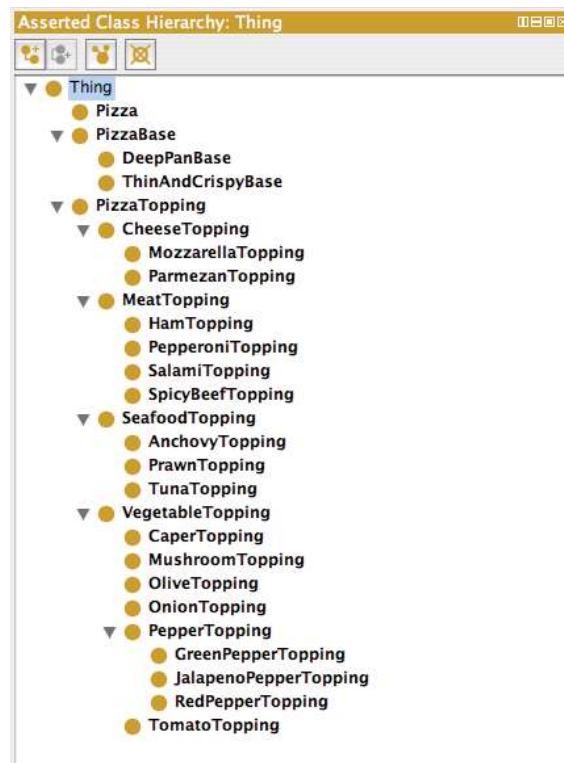
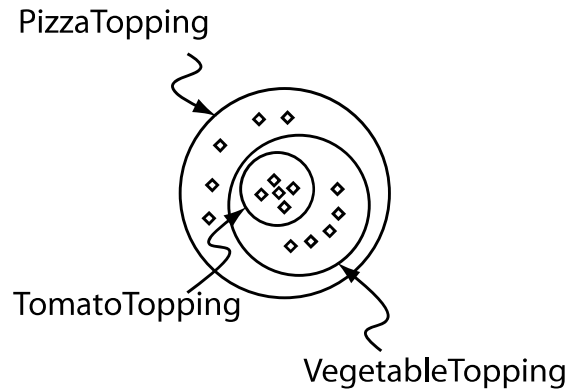


Figure 4.10: Class Hierarchy



**Figure 4.11:** The Meaning Of Subclass — *All* individuals that are members of the class **TomatoTopping** are members of the class **VegetableTopping** and members of the class **PizzaTopping** as we have stated that **TomatoTopping** is a subclass of **VegetableTopping** which is a subclass of **PizzaTopping**

#### MEANING



Up to this point, we have created some simple named classes, some of which are *subclasses* of other classes. The construction of the class hierarchy may have seemed rather intuitive so far. However, what does it actually mean to be a *subclass* of something in OWL? For example, what does it mean for **VegetableTopping** to be a *subclass* of **PizzaTopping**, or for **TomatoTopping** to be a *subclass* of **VegetableTopping**? In OWL *subclass* means *necessary implication*. In other words, if **VegetableTopping** is a *subclass* of **PizzaTopping** then *ALL* instances of **VegetableTopping** are instances of **PizzaTopping**, *without exception* — if something is a **VegetableTopping** then this *implies* that it is also a **PizzaTopping** as shown in Figure 4.11.<sup>a</sup>

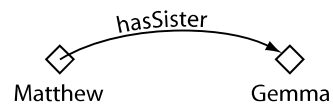
<sup>a</sup>It is for this reason that we seemingly pedantically named all of our toppings with the suffix of ‘Topping’, for example, **HamTopping**. Despite the fact that class names themselves carry no formal semantics in OWL (and in other ontology languages), if we had named **HamTopping** **Ham**, then this could have implied to human eyes that anything that is a kind of ham is also a kind of **MeatTopping** and also a **PizzaTopping**.

## 4.4 OWL Properties

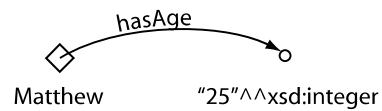
OWL Properties represent relationships. There are two main types of properties, *Object properties* and *Datatype properties*. Object properties are relationships between two individuals. In this chapter we will focus on *Object properties*; *datatype properties* are described in Chapter 5. Object properties link an individual to an individual. OWL also has a third type of property – *Annotation properties*<sup>3</sup>. Annotation properties can be used to add information (metadata — data about data) to classes, individuals and object/datatype properties. Figure 4.12 depicts an example of each type of property.

Properties may be created using the ‘**Object Properties**’ tab shown in Figure 4.13. Figure 4.14 shows the buttons located in the top left hand corner of the ‘**Object Properties**’ tab that are used for creating

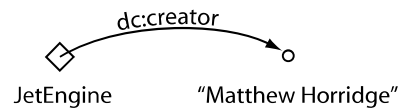
<sup>3</sup>Object properties and Datatype properties may be marked as Annotation properties



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.



An annotation property, linking the class 'JetEngine' to the data literal (string) "Matthew Horridge".

**Figure 4.12:** The Different types of OWL Properties

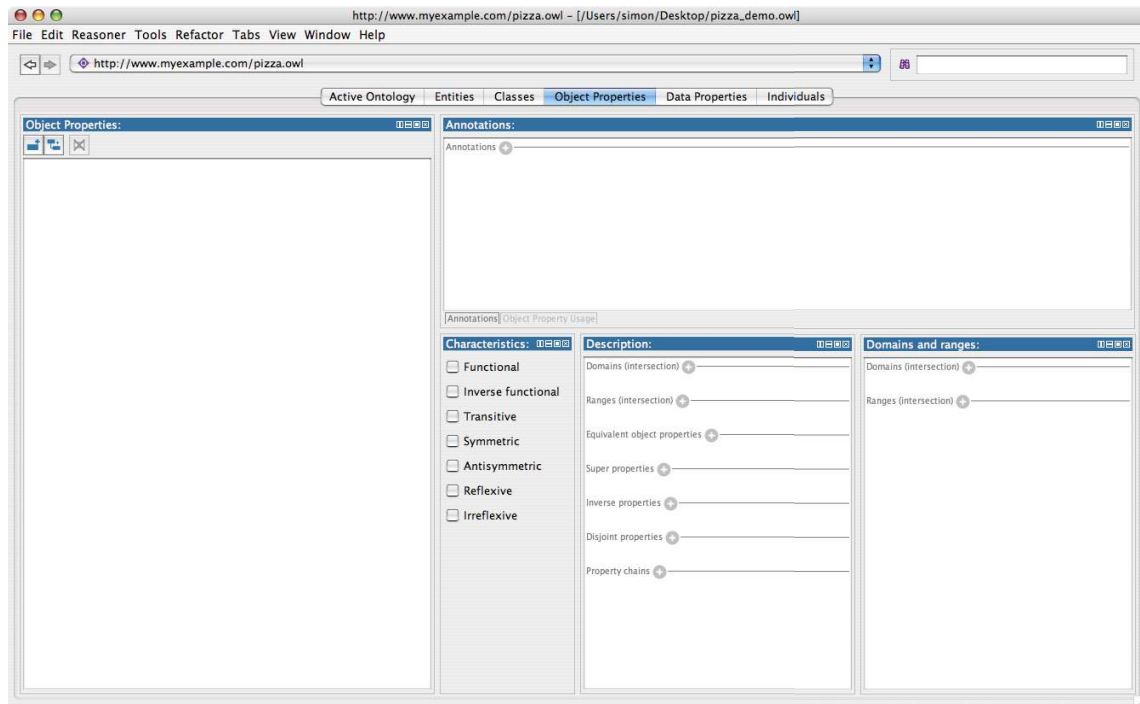
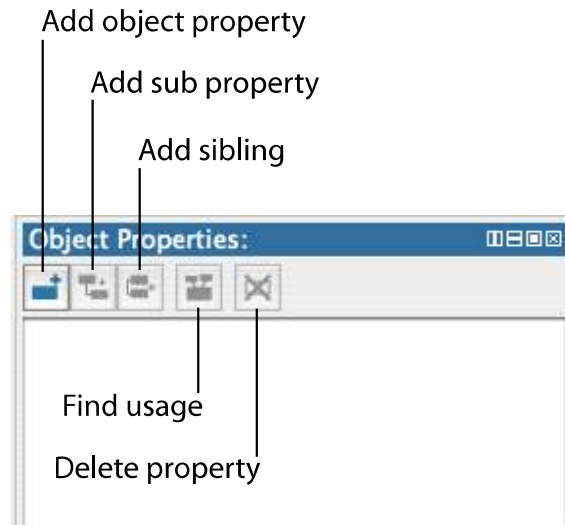


Figure 4.13: The PropertiesTab

OWL properties. As can be seen from Figure 4.14, there are buttons for creating Datatype properties, Object properties and Annotation properties. Most properties created in this tutorial will be **Object properties**.

#### Exercise 8: Create an object property called hasIngredient

1. Switch to the '**Object Properties**' tab. Use the '**Add Object Property**' button (see Figure 4.14) to create a new Object property.
2. Name the property to **hasIngredient** using the '**Property Name Dialog**' that pops up, as shown in Figure 4.15 (The '**Property Name Dialog**').



**Figure 4.14:** Property Creation Buttons — located on the Properties Tab above the property list/tree



**Figure 4.15:** Property Name Dialog

## TIP

Although there is no strict naming convention for properties, we recommend that property names start with a lower case letter, have no spaces and have the remaining words capitalised. We also recommend that properties are prefixed with the word ‘has’, or the word ‘is’, for example **hasPart**, **isPartOf**, **hasManufacturer**, **isProducerOf**. Not only does this convention help make the intent of the property clearer to humans, it is also taken advantage of by the ‘English Prose Tooltip Generator’<sup>a</sup>, which uses this naming convention where possible to generate more human readable expressions for class descriptions.

<sup>a</sup>The English Prose Tooltip Generator displays the description of classes etc. in a more natural form of English, making it easy to understand a class description. The tooltips pop up when the mouse pointer is made to hover over a class description in the user interface.

Having added the **hasIngredient** property, we will now add two more properties — **hasTopping**, and **hasBase**. In OWL, properties may have sub properties, so that it is possible to form hierarchies of properties. Sub properties specialise their super properties (in the same way that subclasses specialise their superclasses). For example, the property **hasMother** might specialise the more general property of