**Chapter 2:**

Start up Protégé. Once Protégé has been started for the first time, enable the additional tabs that will be necessary for working with an ontology. Enable the Classes, Object Properties, Data Properties, and Annotation Properties tabs by locating the options under the Window-> Tabs menu.

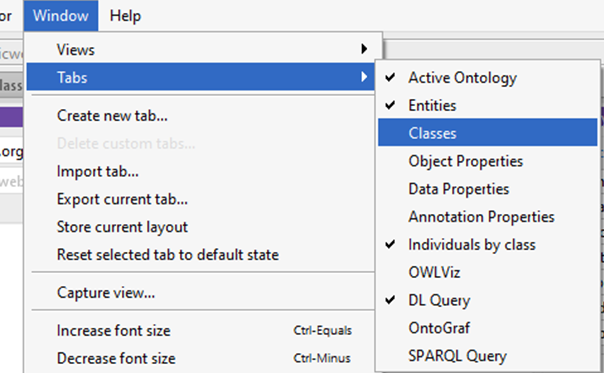


Figure : Tabs Menu

Return to the Active Ontology tab and replace the default URI with your preferred webspace or an example workspace.

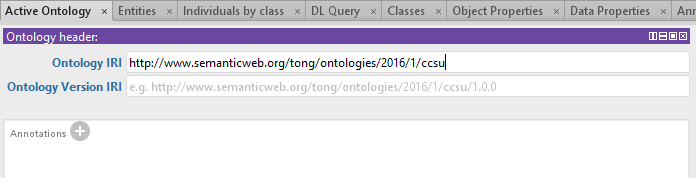


Figure : Setting URI

Now move to the Classes tab. This window will allow us to create new classes for our hierarchy.

* Class Hierarchy view will allow us to create subclasses.
* Annotations view will allow us to create comments and information about the classes.
* Description view will allow us to describe the properties of the subclasses.

The Class Hierarchy view has three buttons, add subclass, add sister class, and delete selected class. All classes are subclass of Thing.

We will begin our ontology by creating the University subclass. Highlight the class you want to make a subclass under and press the add subclass button.

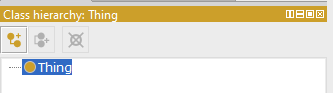


Figure : Class Hierarchy

In the following dialog box, input the class name and press OK to create the subclass.

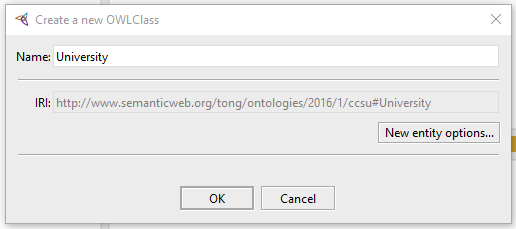


Figure : Class Creation Dialog Box

This will result in the University subclass being made as a subclass of Thing. Go ahead and make additional subclasses. Ensure that you highlight the correct class and use the Add Subclass (if creating a subclass of the highlighted class) or Add Sibling class button (if creating a class on the same level). Make additional classes similar to the screenshot.

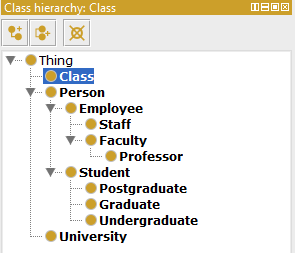


Figure : Sample Class Hierarchy

Next we will disjoint the Employee and Student classes. Two classes are disjoint if individuals cannot be an instance of both classes. Begin by selecting the Professor class. Then click the plus sign next to the "Disjoint With" in the Description window. This will bring up the dialog box. Select the student class and press OK to disjoint the two classes.

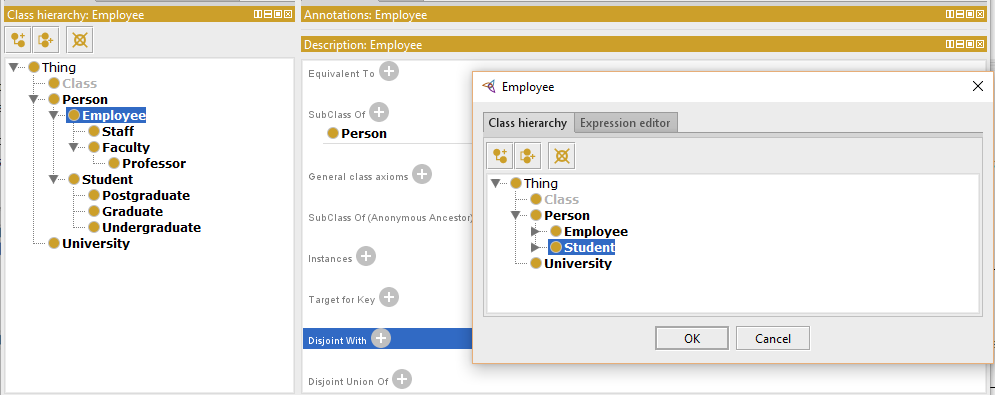


Figure : Disjointing Classes

Let's set some OWL properties in Protégé. Properties or relationship are broken into Object and Datatype properties. In terms of triple-stores properties would be equivalent to predicates.

Switch to the Object Properties view (remember you must enable this view first by going to Window-> Tabs->Object Properties). Familiarize yourself with this view; you will notice the similarities with the Classes view.

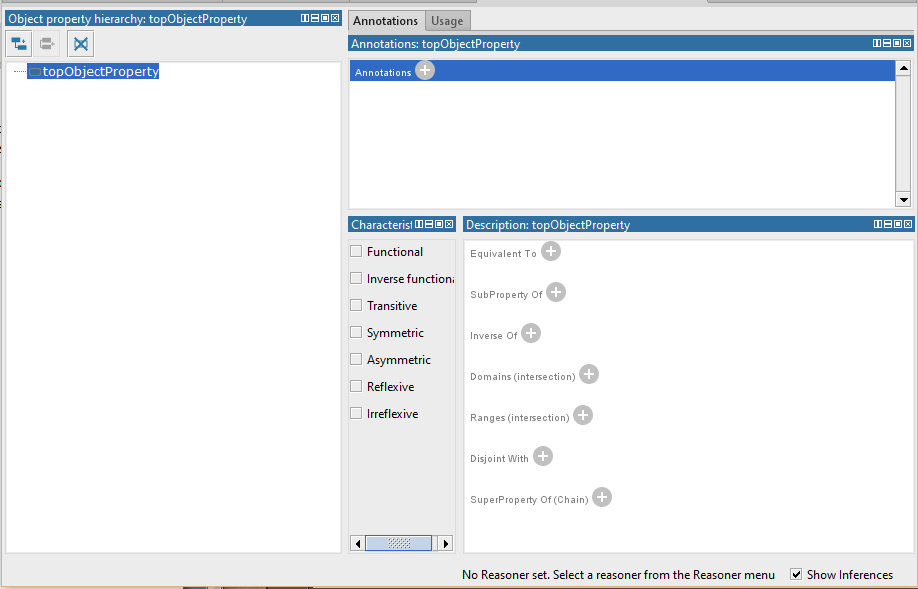


Figure : Object Property Tab

Click on the "Add sub property" button to bring up the dialog to create a new property. Type in "hasEmployee" and press Ok to create the property.

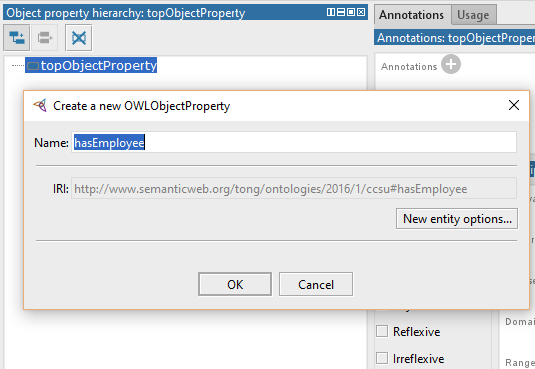


Figure : Creating a new Property

We can also create an Inverse property which corresponds with an object property. For instance we can create the "isEmployedBy" property which is the inverse of "hasEmployee". Select the isEmployedBy property and click on the plus sign next to "Inverse Of" in the Description view. Select the hasEmployee property and press OK.

Go ahead and create additional properties such as "hasStudent" and "isTaughtBy". These additional properties will be useful later in our example.

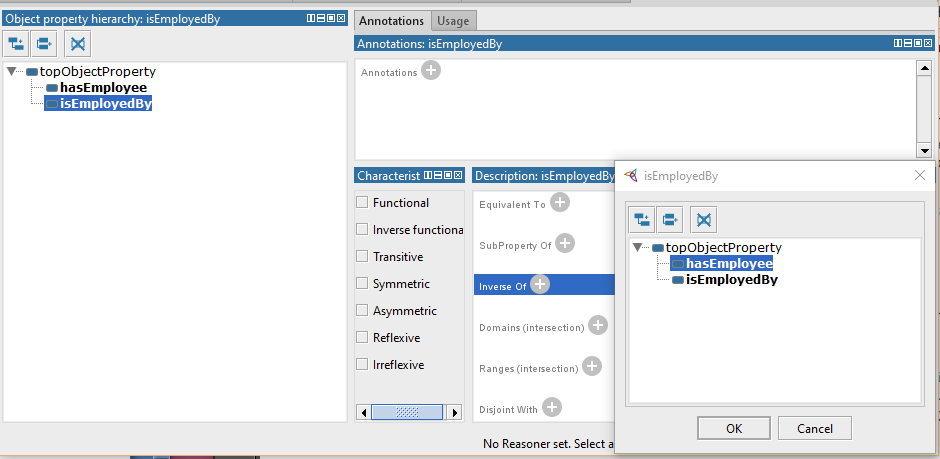


Figure : Setting Inverse Property

See the screenshot below to see what the Description View after an inverse property has been set.

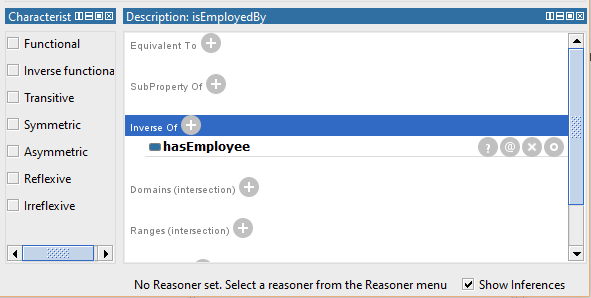


Figure : Inverse Property

Object properties have characteristics that may be set. We will describe some of the characteristics:

* Functional properties have at most one individual related to another individual by the property. For instance "hasBirthMother" would be a functional property.
* Inverse Functional properties describe the property related to the functional property. For instance "isBirthMotherOf" would be the inverse functional property of "hasBirthMother".
* Transitive properties state a related to b, b related to c, therefore a related to c.
* Symmetric Properties state a related to b therefore b related to a. For instance "isFriendsWith" would be symmetric as individuals must be friends with each other to the property to be true.
* Asymmetric Properties state a related to b therefore b cannot be related to a. For instance "isEmployedBy" is asymmetric. An employee cannot employ their employer.
* Reflexive properties relate an individual to themselves. For instance "knows" would be reflexive as everyone must be known to themselves.
* Irreflexive properties state a related to b where a and b are not the same.

Characteristics can be set clicking on a property and selecting the appropriate checkboxes.

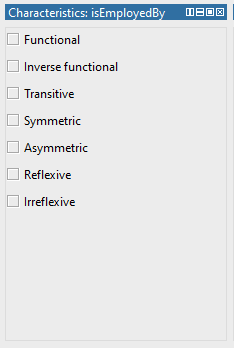


Figure : Characteristics Checkboxes

Properties of a Domain are linked to individuals in a Range. For example in our ontology the property "hasEmployee" has a Range of Employee and Domain University. In effect a domain specifies what subjects a predicate can have while range indicates the objects that the predicate may have.

Let's specify the range of hasEmployee. Select the hasEmployee property and press the plus icon next to "Ranges (intersection)" which will open up a dialog. Click on the Class Hierarchy tab of the dialog and select the Employee class. Press OK to set the range.

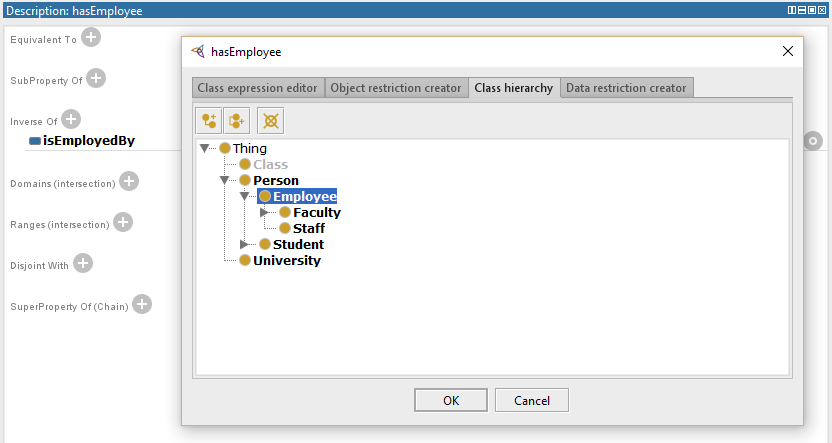


Figure : Add a Range

You can set the domain as the class University using a similar set of steps. The result should be similar to the screenshot below.

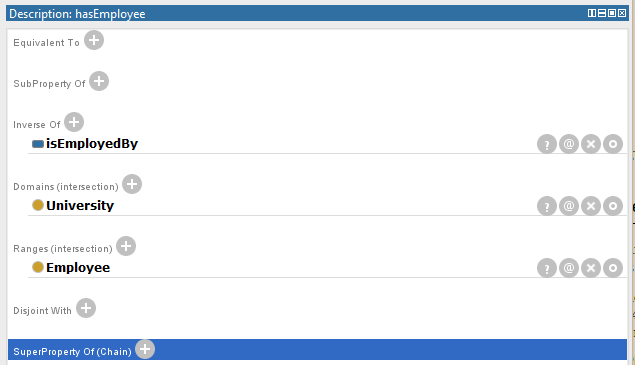


Figure : Adding a Domain

Classes may be restricted by properties. These restrictions can be used to clarify the class. For instance you can set a restriction such as a Professor must have at least one isEmployedBy relationship.

Restrictions are broken into Quantifier Restrictions, Cardinality Restrictions, and hasValue restrictions for OWL. Quantifier Restrictions can also be broken further into Existential and Universal restrictions.

Existential restrictions describe individuals that for a given property have at least one relationship to individuals of a specified class. For instance a Student may have a relationship "isTaughtBy" to the Professor class. To be a student one must have this property with at least one individual but are not limited to just one. These are the most common type of restrictions. Existential restrictions use the keyword "some".

Universal Restrictions describe classes that for a given property only have these relationships to individuals of a specified class. Universal restrictions use the key word "only".

Return to the Classes tab, we will be working with the Description View within this tab. Select the Student class and press the plus icon next to "SubClass Of" in the Description View. This will open up a dialog box. Now switch to the "Object Restriction Creator" tab. Familiarize yourself with this tab. There are three important features; the Restricted Property, Restriction Type, and Restriction Filler features.

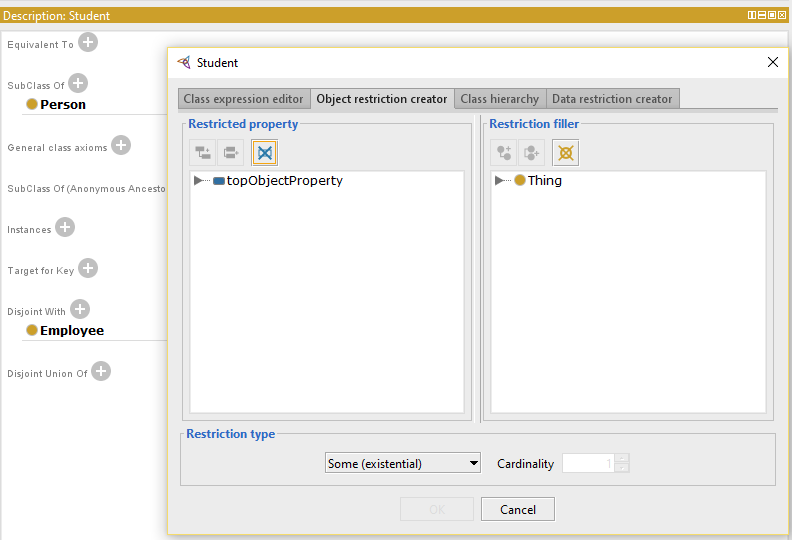


Figure : Creating Restrictions

Now select the isTaughtBy property and Faculty. Make sure the Restrction type is set to "Some" and press OK. This will create a new existential restriction. Note you can create a Universal restriction following the same process and simply by changing the Restriction Type to "Only".

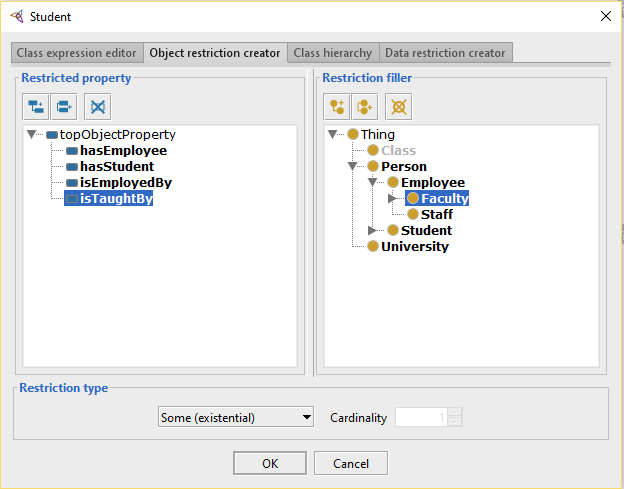


Figure : Restrictions Dialog Box

Alternatively you can use the Class Expression Editor tab to create restrictions. This tab is more powerful but more prone to error. Typing in "isTaughtBy some Faculty" and pressing ok will create the existential restriction. Notice that Protégé will underline what you type until the expression is valid.

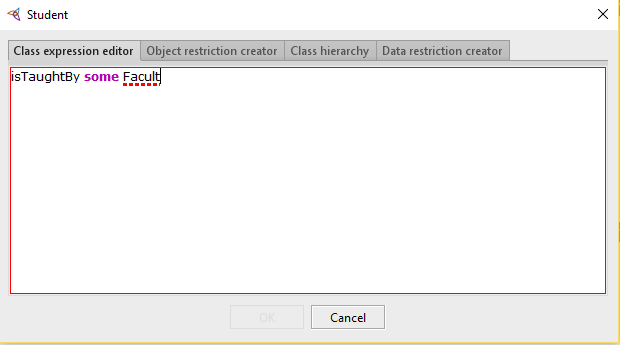


Figure : Class Expression Editor

See the screenshot below to see a completed restriction.

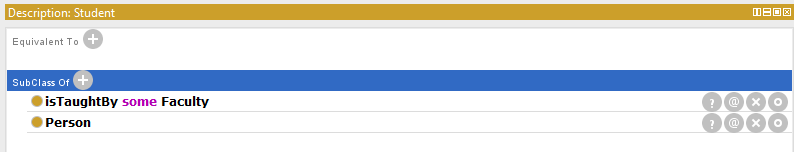


Figure : Completed Restrictions

Next let us use the Protégé Reasoner to test our ontology. For Protégé 5.0; go to Reasoner and select HermiT as our Reasoner. Then go to Reasoner and select Start Reasoner to start. If at any time you make a change to the classes or properties, you can select Synchronize Reasoner to re-perform the check.

The Reasoner will highlight any classes that are inconsistent in red. If nothing is inconsistent the class hierarchy will stay black.

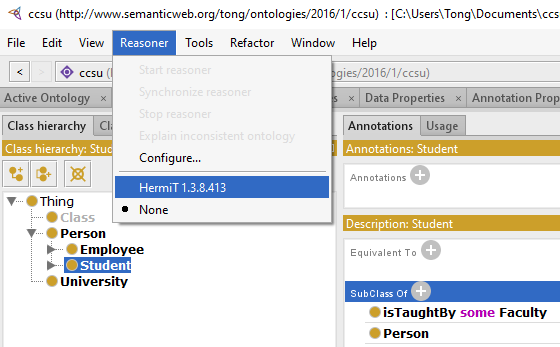


Figure : Reasoner Menu

You can also create probe classes to test restrictions. Probe classes are made purposefully inconsistent with the ontology to test if a restriction properly flags it.

So far we have worked with Primitive classes but there also exists Defined Classes. A Primitive class says that a class must fulfill a necessary condition. A define class says that if a classes meets some conditions then it is sufficient to define this class.

Let us select the Employee class and press the plus icon next to the "SubClass Of" in the Description View. Set the restriction to "isEmployedBy some University". Now select the newly created restriction.



Figure : Complete Restriction

Go to Edit-> Convert to a defined class to change.

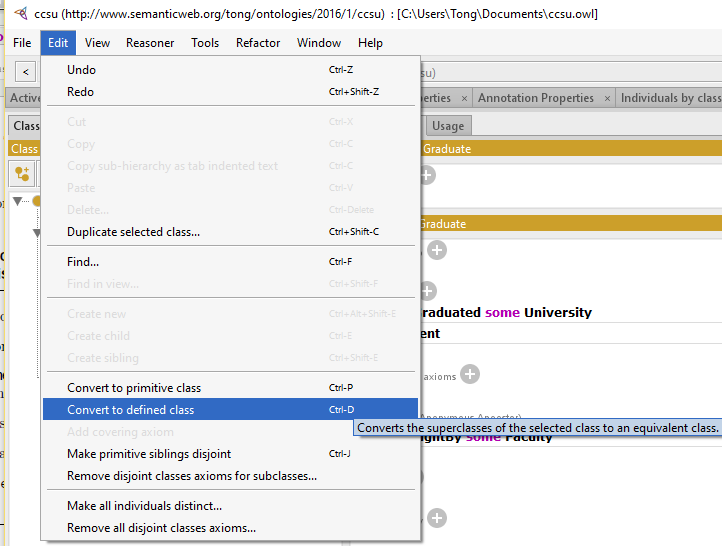


Figure : Defined Class Menu

This will turn the Graduate class into a defined class. Defined classes are marked by the icon featuring three dash lines. Now any individual that has graduated from a University will be placed under the Graduate class. Realistically you would need additional restrictions before classifying an individual as a Graduate such as admission to a Graduate Program.

Go to the Reasoner and select Synchronize to automatically place any classes that meet the restrictions into a defined class. Switch to the "Class Hierarchy (inferred)" to view the inferred classifications.

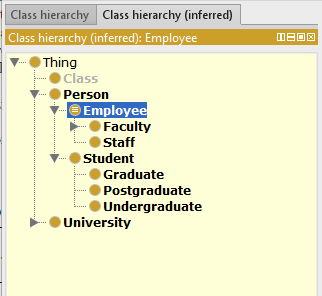


Figure : Inferred Class Hierarchy

It is important to note that OWL is based on open world assumptions also know as open world reasoning. This means that until something is stated to be true it cannot be assumed to be false.

A closure axiom on a universal property says that it can only be filled by the specified fillers. In order for classes to be automatically set you must provide a closure axiom.

Cardinality restrictions can be used to describe a property that has keywords "at least", "at most" or "exactly" specified number of relationships. For example we can create an "isTakingClass" property. Then we can define that a Student must be taking "at least" one class.

Select the student class and press the plus icon next to "SubClass Of". In the Object Restriction Creator choose the appropriate property and filler. In the Restriction Type select "Min (min cardinality)" and set the Cardinality to "1". Press Ok to create the restriction. Follow the same process to create any other cardinality restriction simply by changing the Restriction Type.

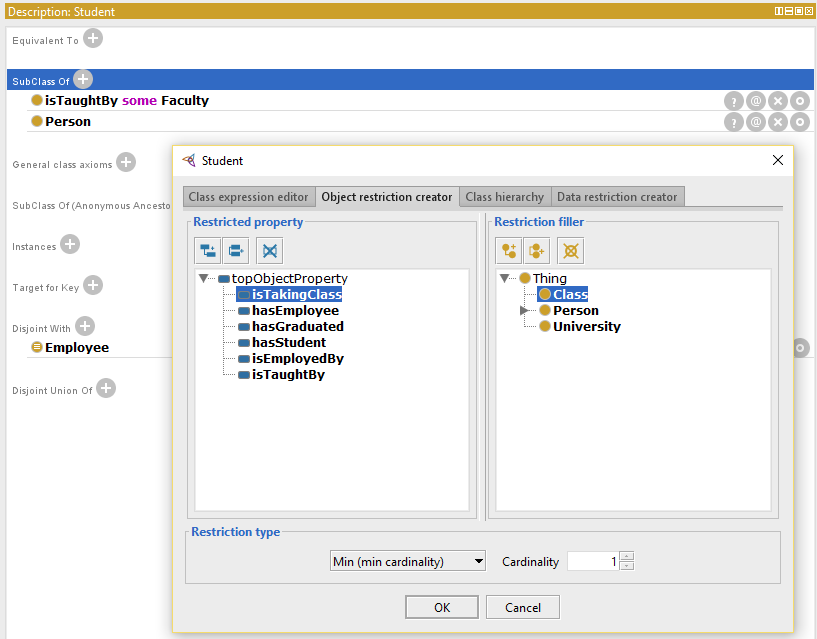


Figure : Cardinality Example

Now let us create some Datatype Properties and Individuals. First switch to the "Data Properties" Tab (enable it in the Window -> Tabs menu if it is missing). You'll be greeted by a familiar but slightly different interface. Click on the Add sub property icon and create a new Data Property called "hasGPAValue".

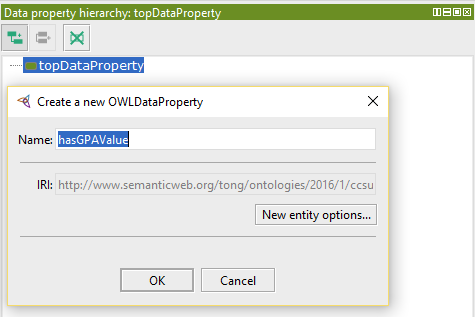


Figure : Creating Data Properties

Now switch to the Individual by Class tab (remember to enable this tab if necessary). Let us create an individual of class undergraduate named Timmy. Select the undergraduate class and click on the "Add Individual" button (has a diamond icon). Create the individual "Timmy".

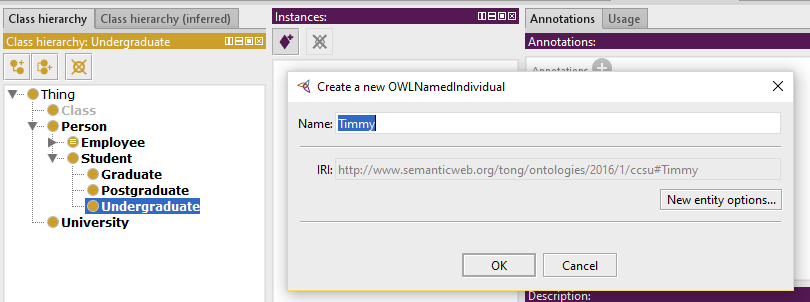


Figure : Creating an Instance

Click on the plus sign next to "Data Property Assertions" in the Property Assertions View to pull up the dialog box. Select the "hasGPAValue" data property. Choose Type Double and press OK to create the data property.

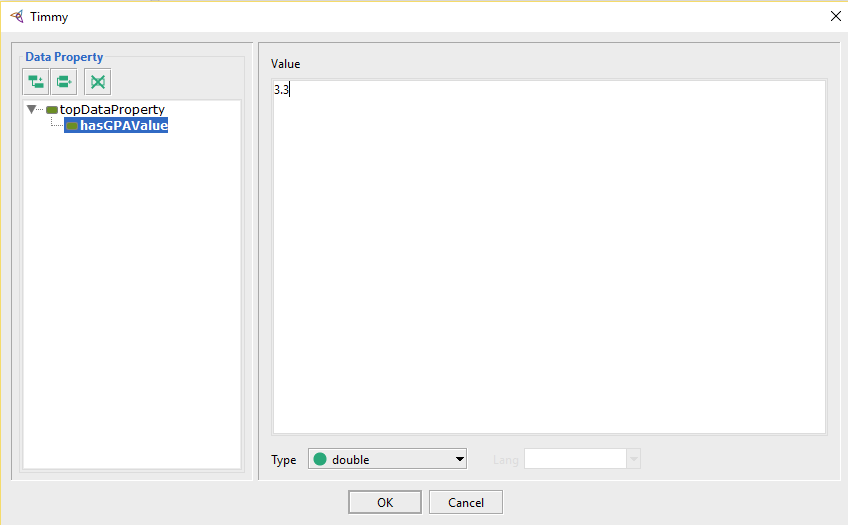


Figure : Data Property Dialog Box

Protégé also allows for the creation of Enumerated Classes. Let us create the "DaysOfTheWeek" class. Go ahead and create the subclass. Then switch over to the Individuals by Class tab. Create an individual for each day of the week. See the screenshot for reference.

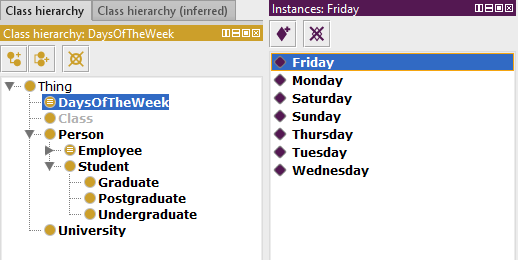


Figure : Creating Individual Days

Switch back to the Classes tab. Click on the plus sign next to "Equivalent To" to bring up the dialog. Change to the Class Expression Editor tab and type "Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}" and press OK. This will set the individuals as an enumeration of the DaysOfTheWeek class.

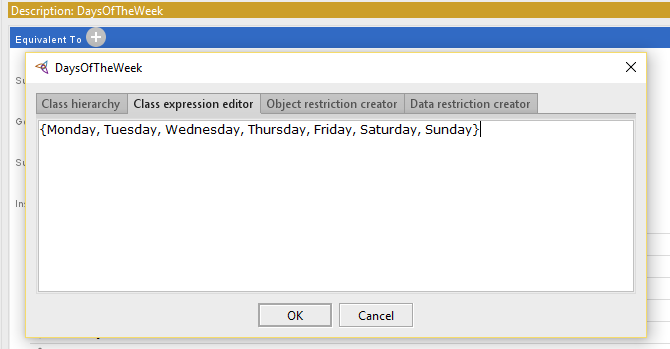


Figure : Enumeration Creation