

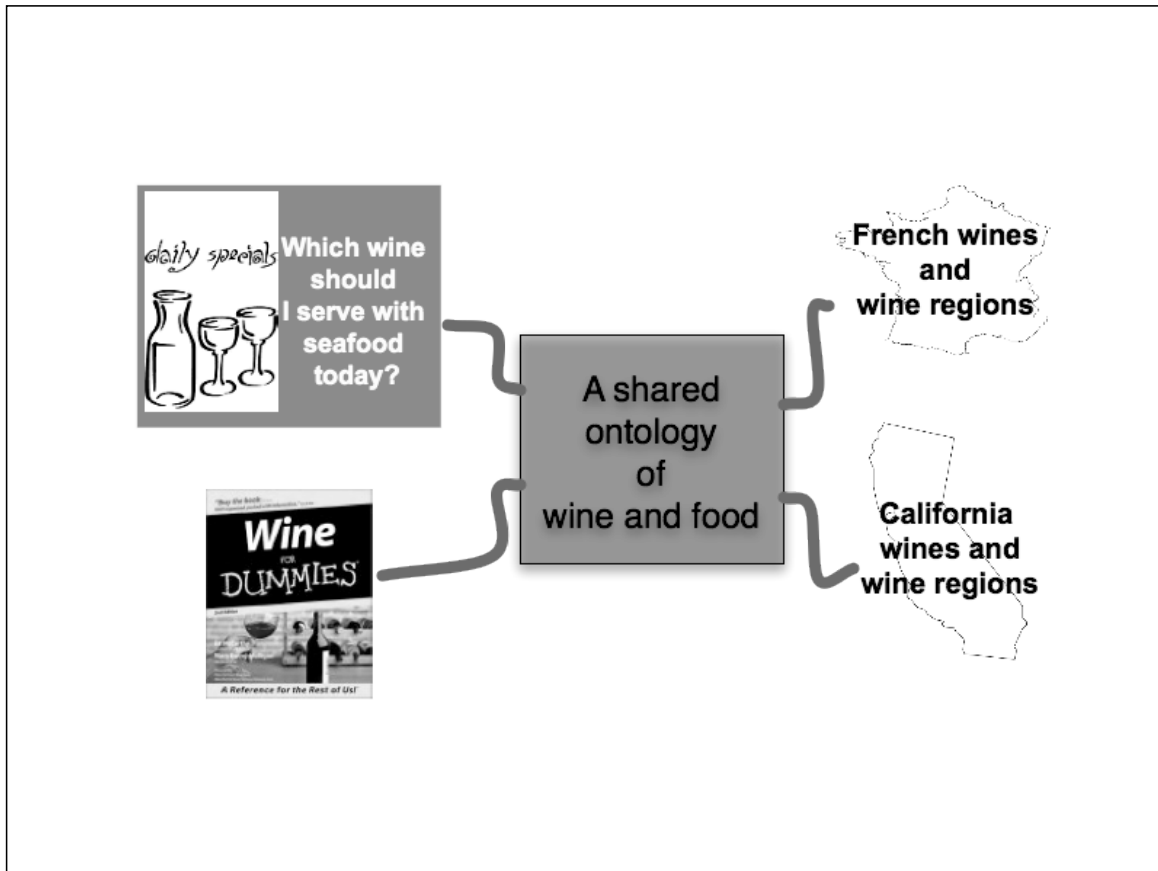
Ontology Development 101

*Natasha Noy
Stanford University*

*A large part of this tutorial is based on
“Ontology Development 101: A Guide to Creating Your First Ontology”
by Natalya F. Noy and Deborah L. McGuinness
http://protege.stanford.edu/publications/ontology_development/ontology101.html*

Outline

- ◆ *What is an ontology?*
 - *definition*
 - *terminology*
- *Why develop an ontology?*
- *Step-By-Step: Developing an ontology*
- *Underwater ??*
 - *What to look out for*



What is an ontology

- *An ontology is an explicit description of a domain:*
 - *concepts*
 - *properties and attributes of concepts*
 - *constraints on properties and attributes*
 - *individuals*
- *An ontology defines*
 - *a common vocabulary*
 - *a shared understanding*

Ontology examples

- *Taxonomies on the Web*
 - *Yahoo! categories*
- *Catalogs for on-line shopping*
 - *Amazon product catalog*
- *Domain-specific standard terminology*
 - *Unified Medical Language System (UMLS)*
 - *UNSPSC - terminology for products and services*

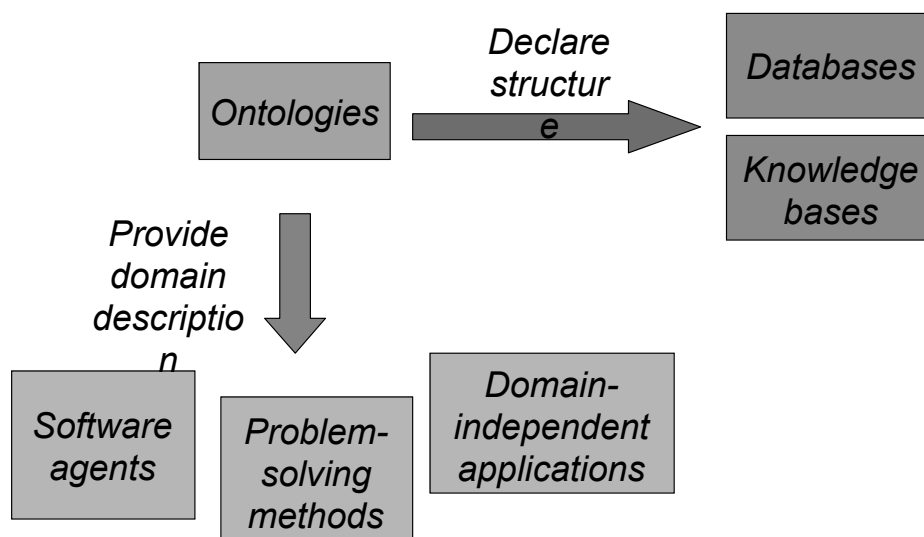
Why develop an ontology?

- *To share common understanding of the structure of information*
 - *among people*
 - *among software agents*
- *To enable reuse of domain knowledge*
 - *to avoid “re-inventing the wheel”*
 - *to introduce standards*

More reasons

- *To make domain assumptions explicit*
 - *easier to change domain assumptions (consider a genetics knowledge base)*
 - *easier to understand and update legacy data*
- *To separate domain knowledge from the operational knowledge*
 - *re-use domain and operational knowledge separately (e.g., configuration based on constraints)*

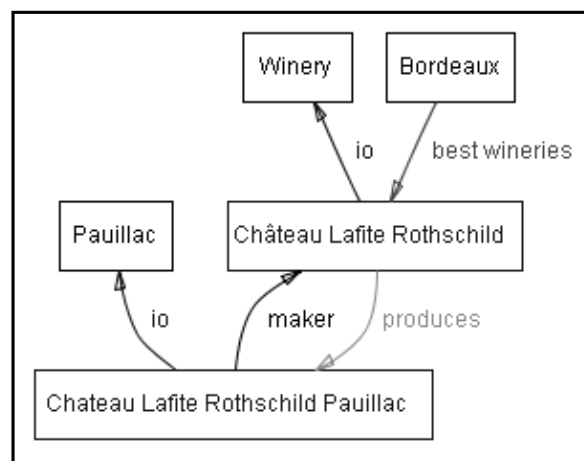
An ontology is often just the beginning



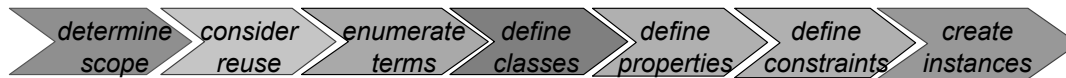
Outline

- *What is an ontology?*
- *Why develop an ontology?*
- ◆ *Step-By-Step: Developing an ontology*
- *Underwater ??*
 - *What to look out for*

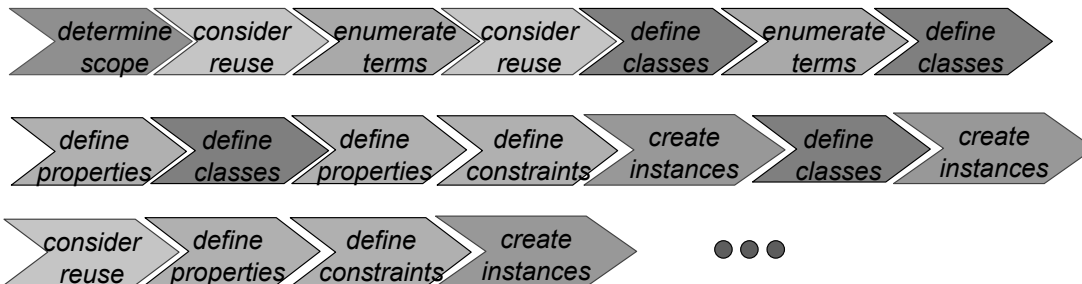
Wines and wineries



Ontology-development process



In reality - an iterative process:



Ontology development versus Object-oriented modeling

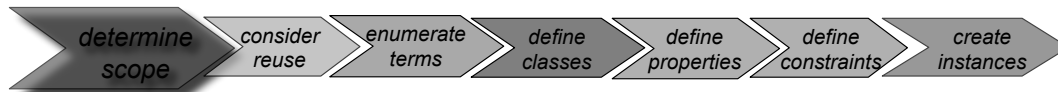
An ontology

- reflects the structure of the world
- is often about structure of concepts
- actual physical representation is not an issue

An OO Structure

- reflects the structure of the data and code
- is usually about behavior (methods)
- describes the physical representation of data (long int, char, etc.)

Determine domain and scope



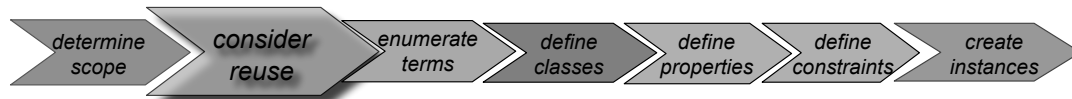
- *What is the domain that the ontology will cover?*
- *For what we are going to use the ontology?*
- *For what types of questions the information in the ontology should provide answers?*
- *Who will use and maintain the ontology?*

Answers to these questions may change during the ontology lifecycle

Competency question for the Wine ontology

- *Which wine characteristics should I consider when choosing a wine?*
- *Is Bordeaux a red or white wine?*
- *Does Cabernet Sauvignon go well with seafood?*
- *What is the best choice of wine for grilled meat?*
- *Which characteristics of a wine affect its appropriateness for a dish?*
- *Does a bouquet or body of a specific wine change with vintage year?*
- *What were good vintages for Napa Zinfandel?*

Consider reuse

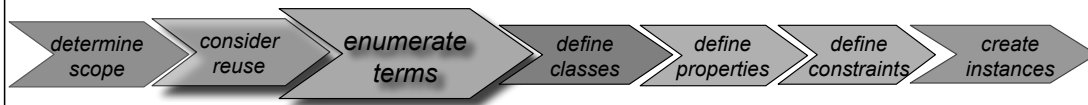


- *Why reuse other ontologies?*
 - *to save the effort*
 - *to interact with the tools that use other ontologies*
 - *to use ontologies that have been validated through use in applications*

What to reuse?

- *Ontology libraries*
 - *Protégé ontology library (protege.stanford.edu)*
 - *Ontolingua ontology library (www.ksl.stanford.edu/software/ontolingua/)*
- *Upper ontologies*
 - *IEEE Standard Upper Ontology (suo.ieee.org)*
 - *Cyc (www.cyc.com)*
- *Domain-specific ontologies*
 - *UMLS Semantic Net*
 - *GO (Gene Ontology) (www.geneontology.org)*
 - *OBO (Open Biological Ontologies) (obo.sourceforge.net)*

Enumerate important terms

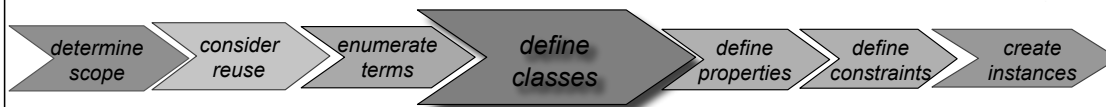


- *What are the terms we need to talk about?*
- *What are the properties of these terms?*
- *What do we want to say about the terms?*

Enumerating terms: The Wine ontology

- *wine, grape, winery, location,*
- *wine color, wine body, wine flavor, sugar content*
- *white wine, red wine, Bordeaux wine*
- *food, seafood, fish, meat, vegetables, cheese*

Define classes and the class hierarchy



- *A class is a concept in the domain*
 - *a class of wines*
 - *a class of wineries*
 - *a class of red wines*
- *A class is a collection of elements with similar properties*
- *Instances of classes*
 - *a glass of California wine you'll have for lunch*

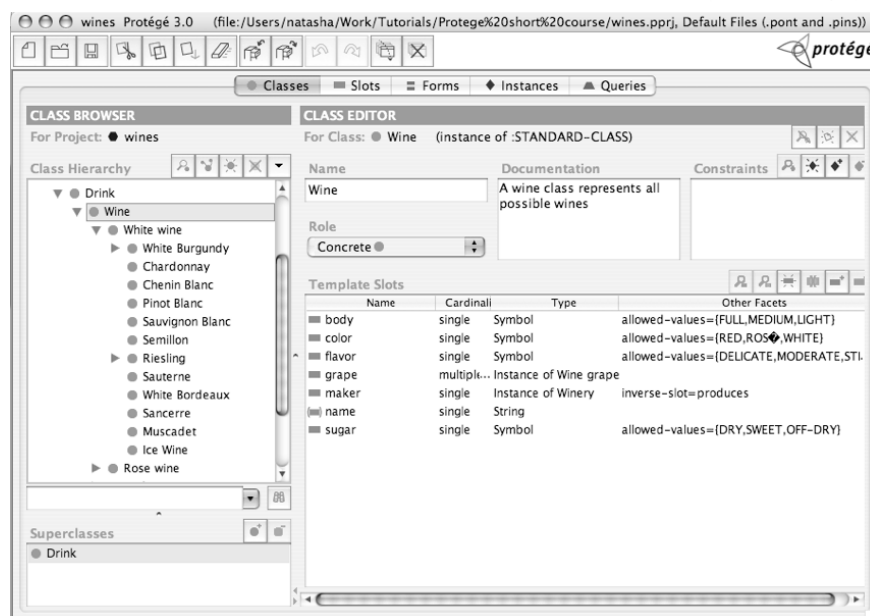
Class inheritance

- *Classes usually constitute a taxonomic hierarchy (a subclass-superclass hierarchy)*
- *A class hierarchy is usually an IS-A hierarchy:*
 - *an instance of a subclass is an instance of a superclass*
- *If you think of a class as a set of elements, a subclass is a subset*

Class inheritance: Examples

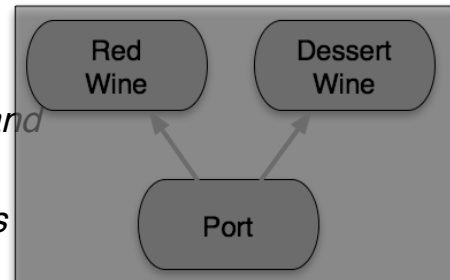
- *Apple is a subclass of Fruit*
 - *Every apple is a fruit*
- *Red wines is a subclass of Wine*
 - *Every red wine is a wine*
- *Chianti wine is a subclass of red wine*
 - *Every Chianti wine is a red wine*

Wine Ontology in Protégé

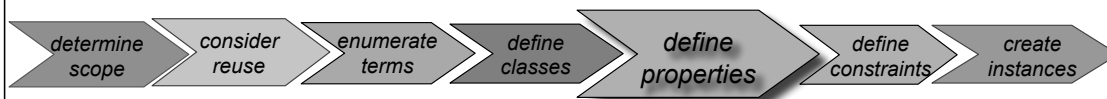


Multiple inheritance

- *A class can have more than one superclass*
- *The subclass inherits properties and restrictions from all the parents*
- *Different systems resolve conflicts differently*



Define properties of classes



- *Slots in a class definition describe attributes of instances of the class*
 - *each wine will have color, sugar content, producer, etc.*

Slots

- *Types of properties*
 - *“intrinsic” properties: flavor and color of wine*
 - *“extrinsic” properties: name and price of wine*
 - *parts: ingredients in a dish*
 - *relations to other objects: producer of wine (winery)*
- *Simple and complex properties*
 - *simple properties (attributes): contain primitive values (strings, numbers)*
 - *complex properties: contain other objects (e.g., a winery instance)*

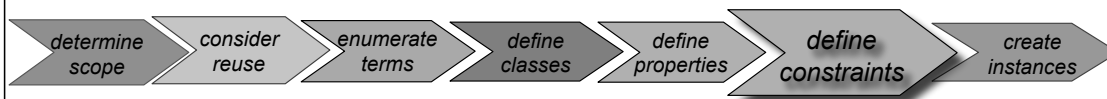
Slots for the class Wine

Template Slots			
Name	Cardinality	Type	Other Facets
body	single	Symbol	allowed-values={FULL,MEDIUM,LIGHT}
color	single	Symbol	allowed-values={RED,ROS,WHITE}
flavor	single	Symbol	allowed-values={DELICATE,MODERATE,STRONG}
grape	multiple...	Instance of Wine grape	
maker	single	Instance of Winery	inverse-slot=produces
name	single	String	
sugar	single	Symbol	allowed-values={DRY,SWEET,OFF-DRY}

Slots and class inheritance

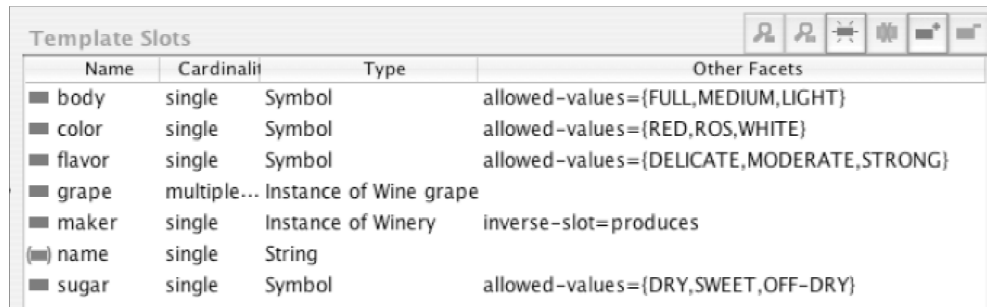
- *A subclass inherits all the slots from the superclass*
 - *If a wine has a name and flavor, a red wine also has a name and flavor*
- *If a class has multiple superclasses, it inherits slots from all of them*
 - *Port is both a dessert wine and a red wine. It inherits “sugar content: high” from the former and “color:red” from the latter*

Property constraints



- *Property constraints (facets) describe or limit the set of possible values for a slot*
 - *the name of a wine is a string*
 - *the wine producer is an instance of Winery*
 - *a winery has exactly one location*

Constraints for the Wine class



Name	Cardinality	Type	Other Facets
body	single	Symbol	allowed-values={FULL,MEDIUM,LIGHT}
color	single	Symbol	allowed-values={RED,ROS,WHITE}
flavor	single	Symbol	allowed-values={DELICATE,MODERATE,STRONG}
grape	multiple...	Instance of Wine grape	
maker	single	Instance of Winery	inverse-slot=produces
name	single	String	
sugar	single	Symbol	allowed-values={DRY,SWEET,OFF-DRY}

Common facets: Cardinality

- *Slot cardinality – the number of values a slot can or must have*
 - *Minimum cardinality*
 - *Minimum cardinality 1 means that the slot must have a value (required)*
 - *Minimum cardinality 0 means that the slot value is optional*
 - *Maximum cardinality*
 - *Maximum cardinality 1 means that the slot can have at most one value (single-valued slot)*
 - *Maximum cardinality greater than 1 means that the slot can have only one value (multiple-valued slot)*

Common facets: Value Type

- *Slot value type – what values can the slot have*
 - *String: a string of characters (“Château Lafite”)*
 - *Number: an integer or a float (15, 4.5)*
 - *Boolean: a true/false flag*
 - *Enumerated type: a list of allowed values (red, white, rosé)*
 - *Complex type: an instance of another class or a class itself*
 - *Specify the class to which the instances belong*
 - *For example, the Wine class is the value type for the produces slot at the Winery class*

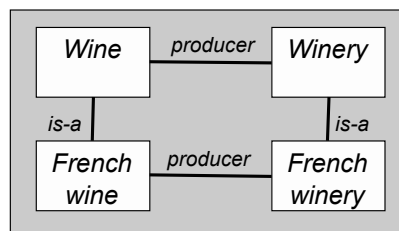
Defining facets: Example

The screenshot shows the 'maker' slot configuration window in the Protege editor. The window title is 'maker (instance of :STANDARD-SLOT)'. It contains several sections for defining the slot's facets:

- Name:** A text field containing 'maker'.
- Value Type:** A dropdown menu set to 'Instance'.
- Allowed Classes:** A list box containing 'Winery'.
- Documentation:** A text area containing 'The maker of a wine (a Winery). This slot has an iinverse – the slot produces at the Winery class'.
- Cardinality:** Two checkboxes, 'required' and 'multiple', both unchecked. The 'at most' value is set to '1'.
- Minimum/Maximum:** Two empty text fields.
- Inverse Slot:** A checkbox labeled 'produces' is checked.
- Template Value:** An empty text field.
- Default Value:** An empty text field.
- Domain:** A list box containing 'Wine'.

Restrictions and class inheritance

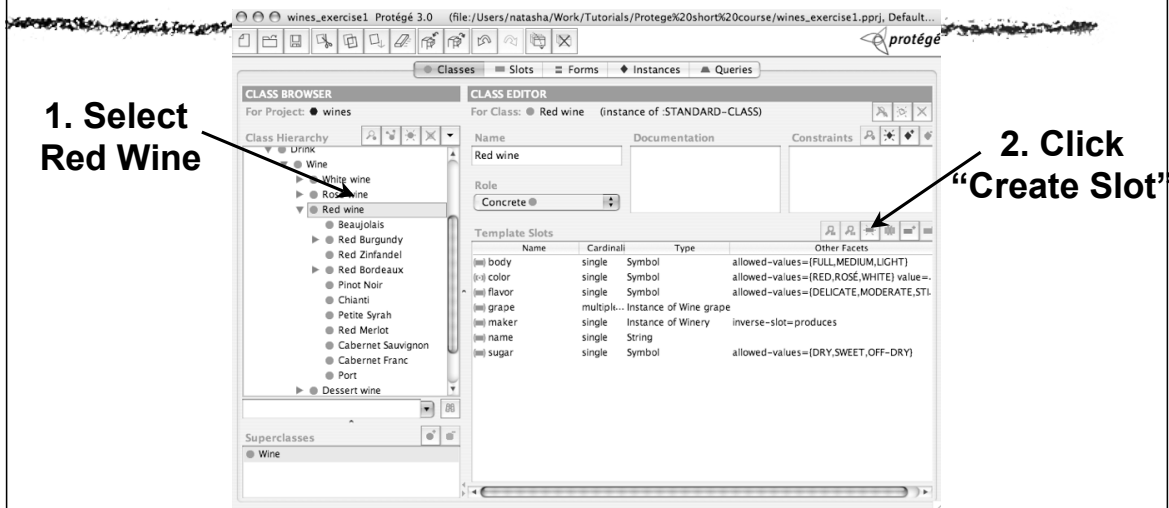
- *A subclass inherits all the slots from the superclass*
- *A subclass can override the restrictions to “narrow” the list of allowed values*
 - *Make the cardinality range smaller*
 - *Replace a class in the range with a subclass*



Global vs Local Constraints

- *Slots are first-class objects*
 - *contrast with fields in OO programming*
- *Global slot constraints apply to the property throughout the ontology*
 - *allowed class for hasColor is always one of the values from the list of colors*
- *Local slot constraints valid only for instances of the class and its subclasses*
 - *hasColor for RedWine has the value Red*

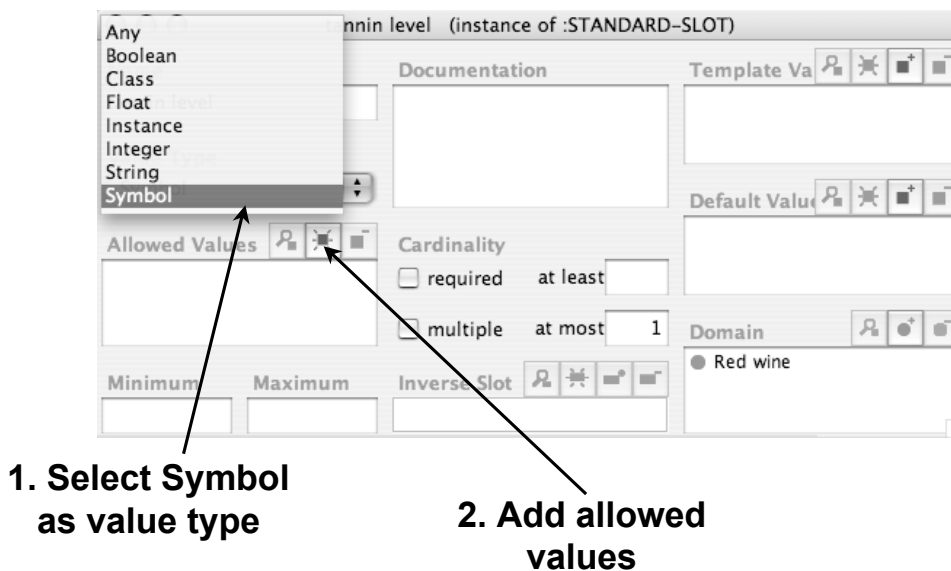
Create a new slot



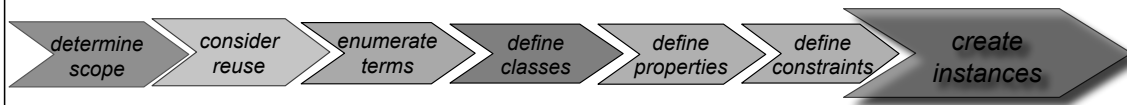
TWO things happened:

- 1. Slot tannin level was created at top level*
- 2. Class Red wine was added to the domain of the slot*

Define Allowed Values



Create instances



- *Create an instance of a class*
 - *The class becomes a direct type of the instance*
 - *Any superclass of the direct type is a type of the instance*
- *Assign property values for the instance frame*
 - *Property values should conform to the constraints*
 - *Knowledge-acquisition tools often check that*

Creating an instance: Example

The screenshot shows a window titled "Chateau Morgon Beaujolais (instance of Beaujolais)". Inside, there is a form with several fields and a list box. The fields are: "Name" (containing "Chateau Morgon Beaujolais"), "Body" (a dropdown menu set to "LIGHT"), "Color" (a dropdown menu set to "RED"), "Flavor" (a dropdown menu set to "DELICATE"), "Sugar" (a dropdown menu set to "DRY"), and "Tannin Level" (a dropdown menu set to "LOW"). To the right of these fields are two list boxes, each with a "Maker" and a "Grape" label. The "Maker" list box contains "Chateau Morgon" and the "Grape" list box contains "Gamay grape". Each list box has a small icon to its right.

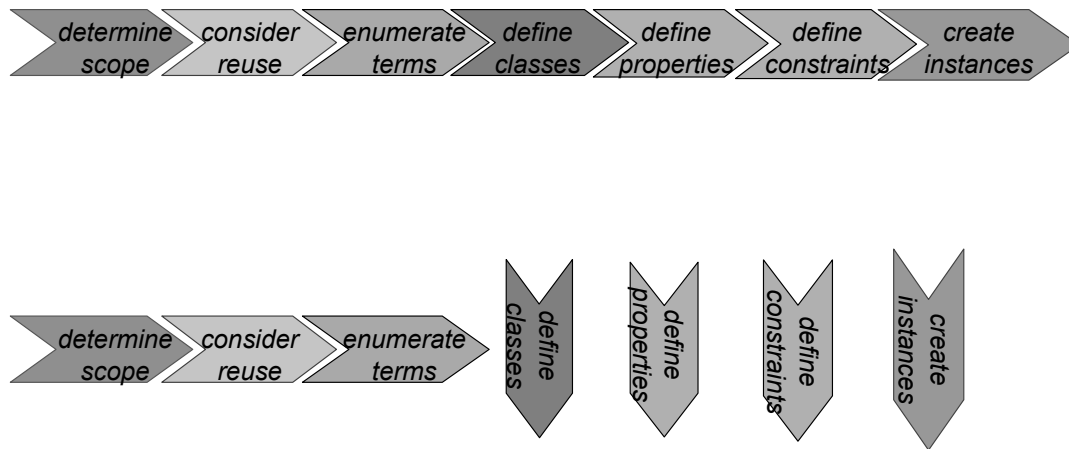
Exercise: Define an instance of your favorite wine

The screenshot shows the Protege editor interface for defining a new instance. The window title is 'Qupé Syrah (instance of Syrah, internal name is wines_exercise...)'. The 'Name' field contains 'Qupé Syrah'. The 'Maker' field has a dropdown menu showing 'Qupé'. The 'Body' field is set to 'FULL'. The 'Color' field is set to 'RED'. The 'Flavor' and 'Sugar' fields are empty. The 'Tannin Level' field is empty. The 'Grape' field is empty. There are icons for adding, deleting, and other actions next to the 'Maker' and 'Grape' fields.

Outline

- *What is an ontology?*
- *Why develop an ontology?*
- *Step-By-Step: Developing an ontology*
- *Underwater ??*
 - *What to look out for*

Going deeper



Going Deeper

- *Defining classes and a class hierarchy*
- *Deciding when to create a new class*
- *Defining constraints on slots*
- *Defining instances (forms)*
- *Naming conventions*
- *General considerations*

Going Deeper

- *Defining classes and a class hierarchy*
- *Deciding when to create a new class*
- *Defining constraints on slots*
- *Defining instances (forms)*
- *Naming conventions*
- *General considerations*

Defining classes and a class hierarchy

- *The question to ask:*
 - *“Is each instance of the subclass an instance of its superclass?”*
- *The things to remember:*
 - *There is no single correct class hierarchy*
 - *But there are some guidelines*

Siblings in the class hierarchy



- *All the siblings in the class hierarchy must be at the same level of generality*
- *Compare to section and subsections in a book*

The perfect family size



- *If a class has only one child, there may be a modeling problem*
- *If the only Red Burgundy we have is Côtes d'Or, why introduce the subhierarchy?*
- *Compare to bullets in a bulleted list*

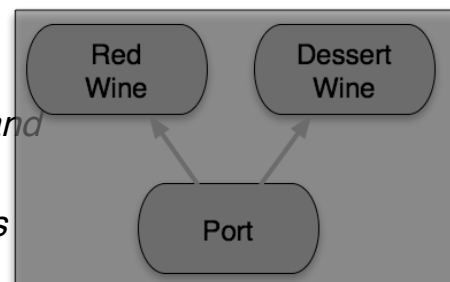
The perfect family size (II)



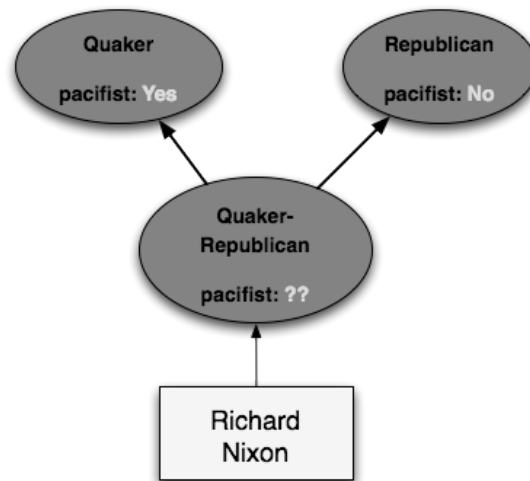
- *If a class has more than a dozen children, additional subcategories may be necessary*
- *However, if no natural classification exists, the long list may be more natural*

Multiple inheritance

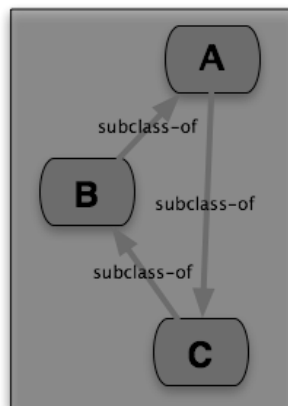
- *A class can have more than one superclass*
- *The subclass inherits properties and restrictions from all the parents*
- *Different systems resolve conflicts differently*



Dangers of multiple inheritance



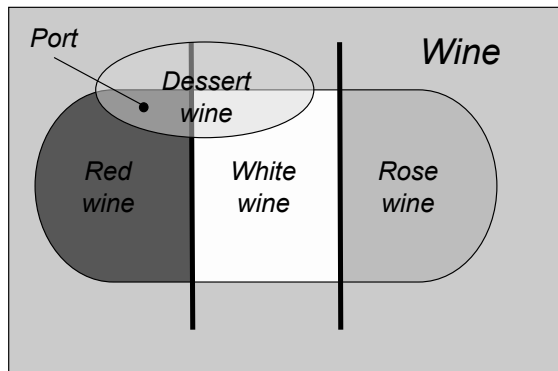
Avoiding class cycles



- *Danger of multiple inheritance: cycles in the class hierarchy*
- *Classes A, B, and C have equivalent sets of instances*
 - *By many definitions, A, B, and C are thus equivalent*

Disjoint classes

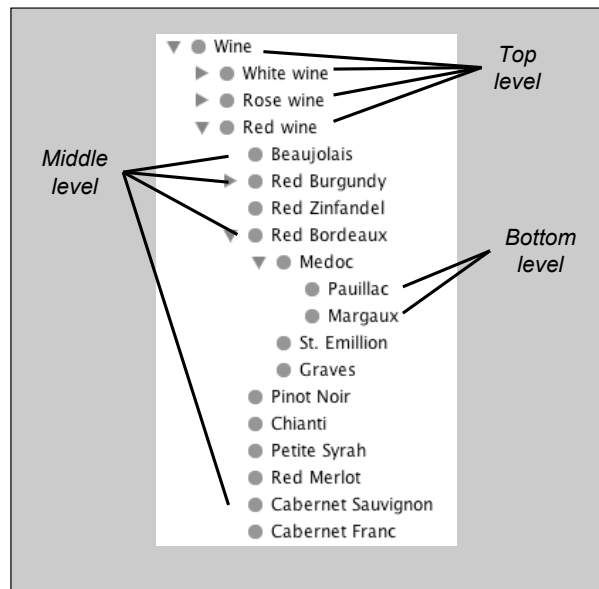
- *Classes are disjoint if they cannot have common instances*
- *Disjoint classes cannot have any common subclasses either*
 - *Red wine, White wine, Rosé wine are disjoint*
 - *Dessert wine and Red wine are not disjoint*



Levels in the class hierarchy

- *Different modes of the development*
 - *top-down - define the most general concepts first and then specialize them*
 - *bottom-up - define the most specific concepts and then organize them in more general classes*
 - *combination*

Levels in the class hierarchy



A completed hierarchy of wines



Going Deeper

- *Defining classes and a class hierarchy*
- *Deciding when to create a new class*
- *Defining constraints on slots*
- *Defining instances (forms)*
- *Naming conventions*
- *General considerations*

Decisions, Decisions....

- *When to introduce a new class?*
 - *Do we need a class for Red Burgundy?*
- *Should a distinction be described by subclasses or property values?*
 - *Class Red Wine versus slot color*
- *Is something a class or an instance?*
 - *Is Riesling a class or an instance?*
- *Should we use enumeration or create a new class?*
 - *Should we have a class for Color or use strings and symbols for values?*

When to introduce a new class?

- *Subclasses of a class usually have*
 - *Additional properties*
 - *Additional restrictions*
 - *Participate in different relationships*
- *Subclasses of a class have*
 - *New properties*
 - *New restrictions*

But

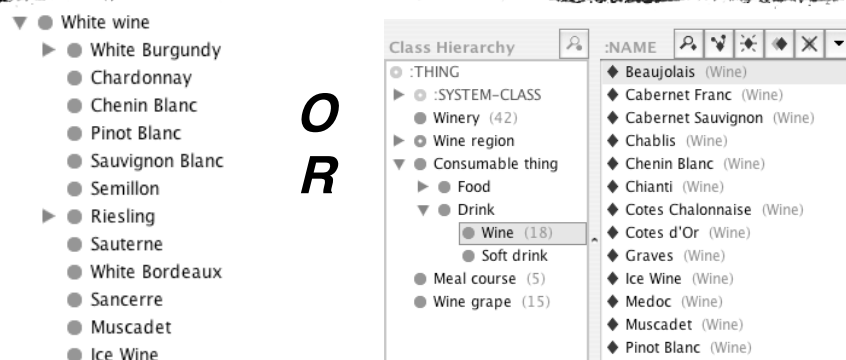
- *In terminological hierarchies, new classes do not have to introduce new properties*

A new class or a property value?



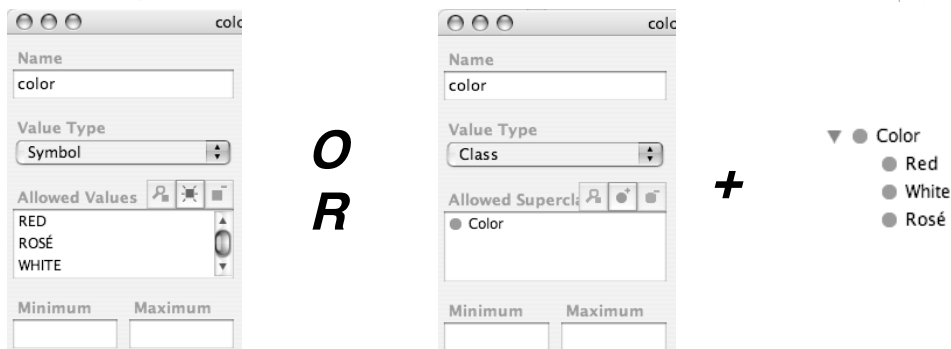
- *Do concepts with different property values become restrictions for different properties?*
- *How important is the distinction for the domain?*
- *A class of an instance should not change often*

A class or an instance?



- *Individual instances are the most specific objects in an ontology*
- *If concepts form a natural hierarchy, represent them as classes*

Enumeration: Classes or Strings?



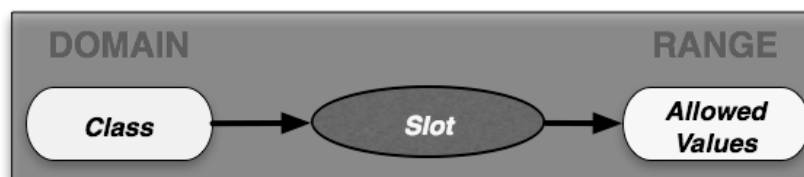
- *Are values used elsewhere in the ontology?*
- *Is there additional information about each value?*
- *Do values form a hierarchy?*

Going Deeper

- *Defining classes and a class hierarchy*
- *Deciding when to create a new class*
- *Defining constraints on slots*
- *Defining instances (forms)*
- *Naming conventions*
- *General considerations*

Back to slots: Domain and range

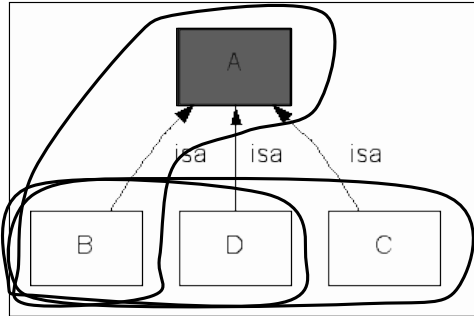
- *Domain of a slot – the class (or classes) that have the slot*
 - *More precisely: class (or classes) instances of which can have the slot*
- *Range of a slot – the class (or classes) to which slot values belong*



Back to slots: Allowed classes

- *When defining a domain or range for a slot, find the most general class or classes*
- *Consider the produces slot for a Winery:*
 - *~~Range: Red wine, White wine, Rosé wine~~*
 - *Range: Wine*
- *Consider the flavor slot*
 - *~~Domain: Red wine, White wine, Rosé wine~~*
 - *Domain: Wine*

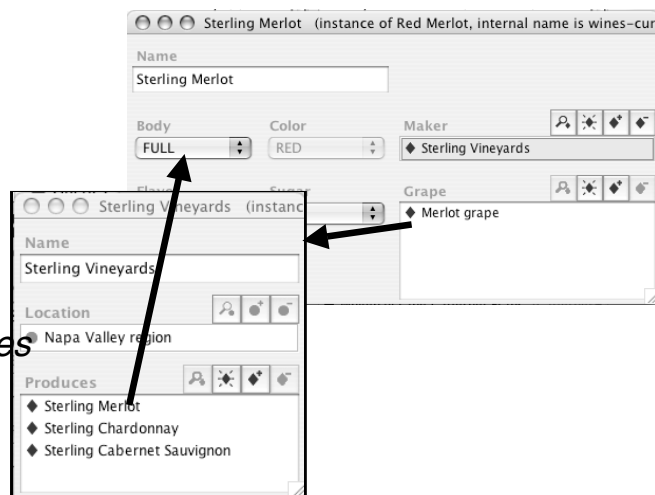
Defining domain and range



- *A class and a superclass – replace with the superclass*
- *All subclasses of a class – replace with the superclass*
- *Most subclasses of a class – consider replacing with the superclass*

Inverse Properties

- *Maker and*
- *Producer*
- *are inverse properties*



Inverse Properties (II)

- *Inverse prorties contain redundant information, but*
 - *Allow acquisition of the information in either direction*
 - *Enable additional verification*
 - *Allow presentation of information in both directions*
- *The actual implementation differs from system to system*
 - *Are both values stored?*
 - *When are the inverse values filled in?*
 - *What happens if we change the link to an inverse property?*

Exercise: Inverse properties

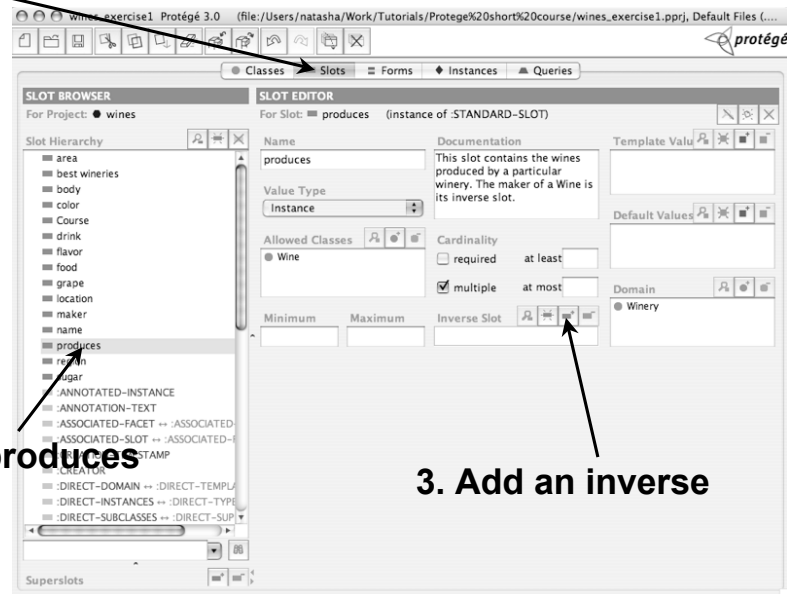
- *Use the opened project wine_exercise1*
- *Declare that slots producer and maker are inverses of each other*
- *Create a new instance of your second favorite wine*
- *Observe what happens to the winery that makes it*

Declare slots to be inverse

1. Select Slots tab

2. Select produces

3. Add an inverse



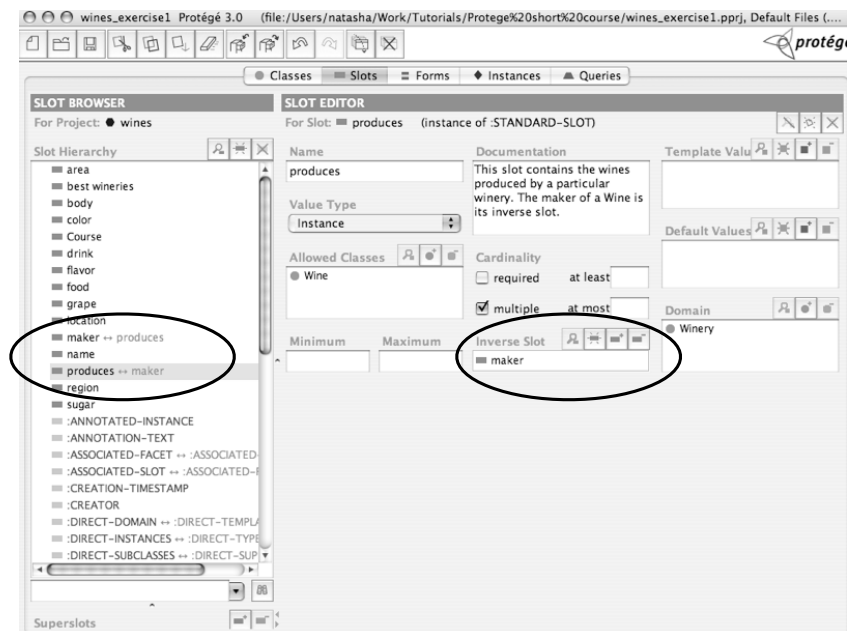
Select and inverse slot

Only slots of type
Instance appear in
the list

*Slots with
primitive value
types cannot have
inverses: there is
no place to put
inverse in*



Resulting inverse slots

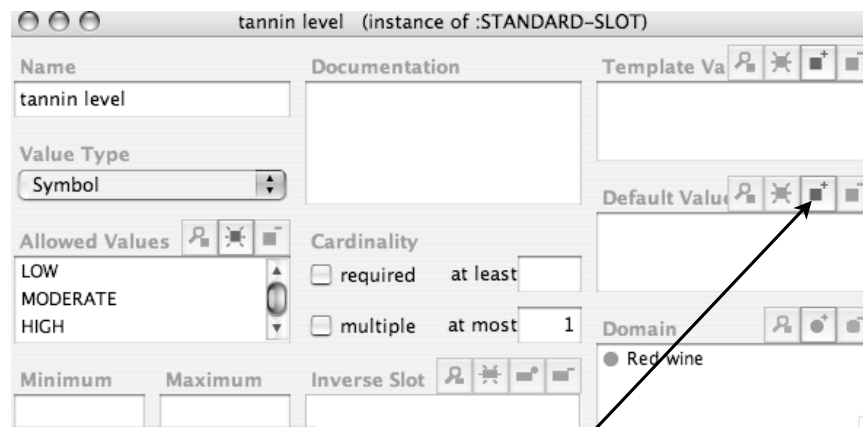


Default values

- *Default value – a value the slot gets when an instance is created*
- *A default value can be changed*
- *The default value is a common value for the slot, but is not a required value*
 - *For example, the default value for wine body can be FULL*

Exercise: creating a default

- *Specify that tannin level is usually medium*



Going Deeper

- *Defining classes and a class hierarchy*
- *Deciding when to create a new class*
- *Defining constraints on slots*
- *Defining instances (forms)*
- *Naming conventions*
- *General considerations*

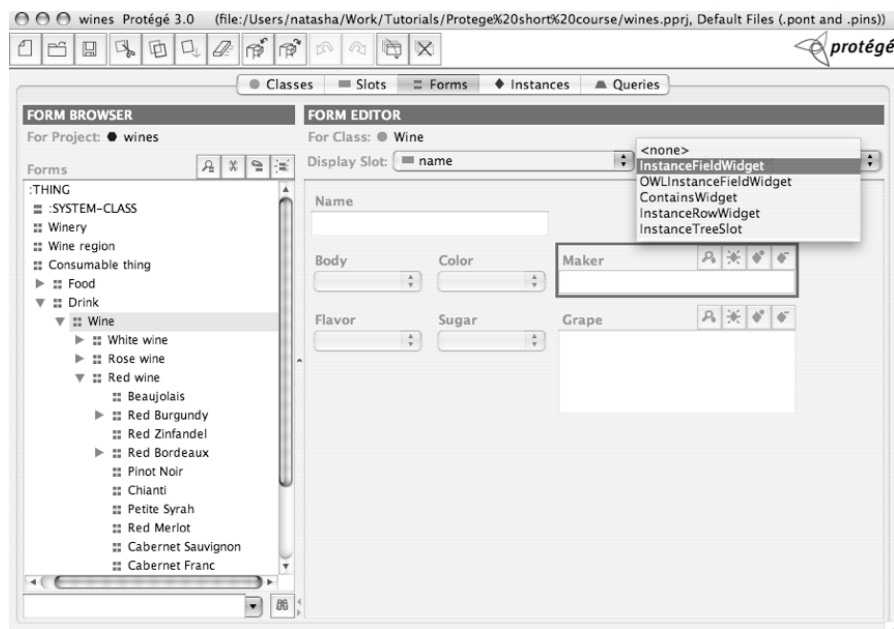
Defining Instances: Form customization

- *Knowledge-acquisition forms (instance forms) are generated automatically*
- *Protege allows customization*
 - *Change the layout of the form*
 - *order, size of the widgets (form components)*
 - *Change labels on the widgets*
 - *Hide slots from the form*
 - *Change widgets used for a slot*

Inheritance

- *Forms are inherited to subclasses*
- *Can override the form at a subclass*
- *Can remove form customization*

Forms



Useful Slot Widgets

- *Contains widget: display another instance inside the current instance*
- *Instance table widget: display several related instances in a table*
- *Image widget: display an image on a form*
- *Graph widget: draw a set of instances as a flow chart (Session on visualization tomorrow)*

Exercise: customizing forms

- *Select the form for class wine*
- *Adjust sizes and positions to your liking*
- *Change labels*
- *Hide slots*

1. Select Forms tab

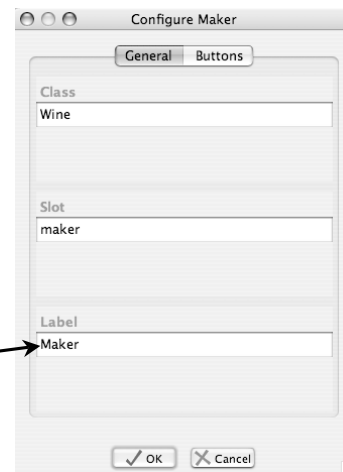
2. Select class Wine

3. Select widgets, resize, move around

Change slot label

- *Double-click on a widget to bring up configuration form for that widget*
- *Change labels and visible buttons*

Type a new label

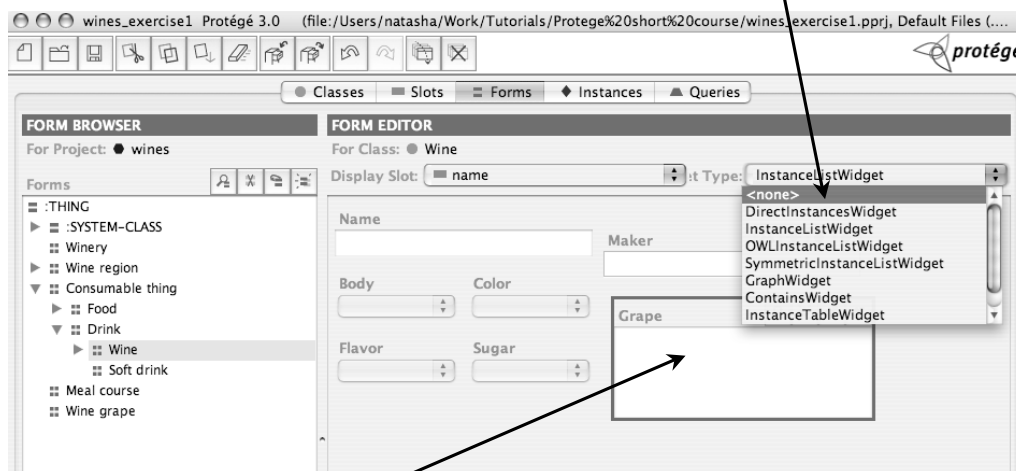


Seeing the results

- *Look at an instance of wine -- it looks different now*

Disabling widgets

2. Select <none> from the list of widgets

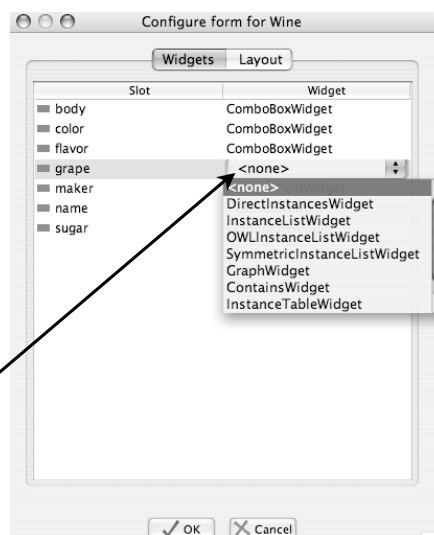


1. Select the widget to disable

Enabling widgets

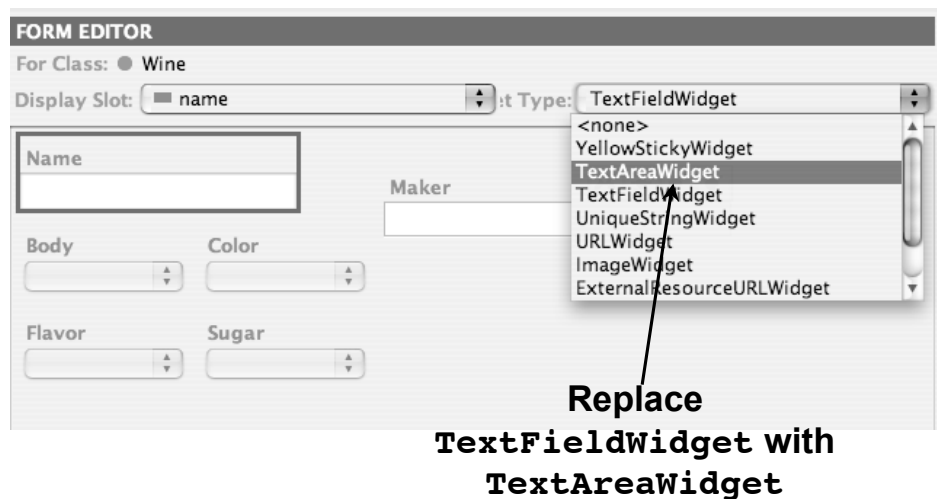
- *Double-click on a blank space on the form to bring up form-customization dialog*

Replace <none> with a new widget



Replacing widget

- *Make a wine name widget into a scrollable multi-line box; resize it*



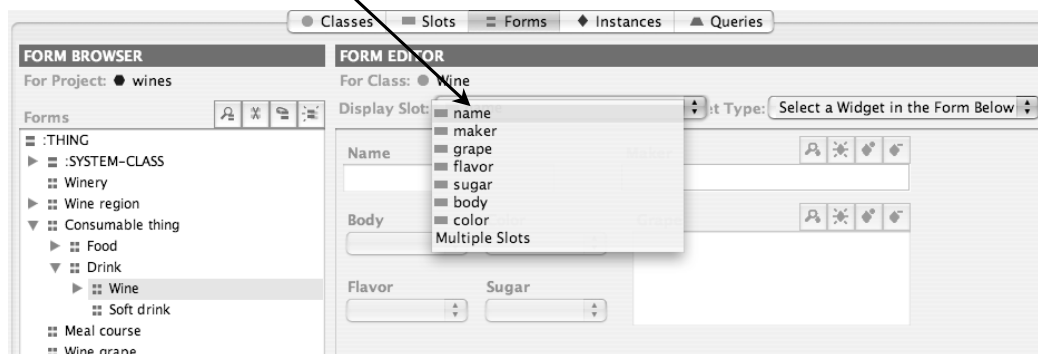
Show winery information on a wine form

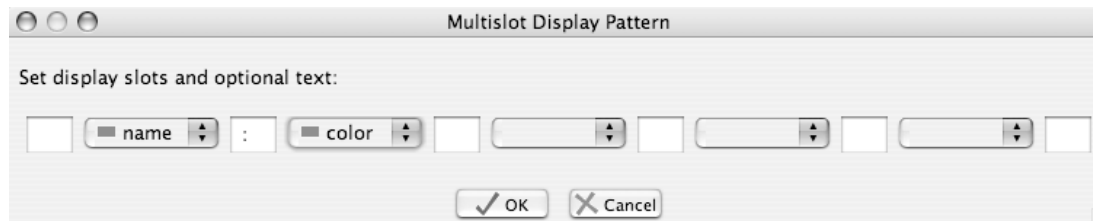
- *Replace widget for maker with ContainsWidget*
- *Check out an instance of a wine*

Display slots

- *Display slots*
 - *determine how instances of a class appear in the UI*
 - *are the same for all direct instances of a class*
- *Defining display slot*
 - *choose a single slot*
 - *choose several slots and put separators*

Choose a display slot





Going Deeper

- *Defining classes and a class hierarchy*
- *Deciding when to create a new class*
- *Defining constraints on slots*
- *Defining instances (forms)*
- *Naming conventions*
- *General considerations*

What's in a name?

- *Define a naming convention for classes and properties and adhere to it*
- *Features of an ontology tool to consider:*
 - *Can classes and properties have the same names?*
 - *Is the system case-sensitive?*
 - *What delimiters are allowed?*

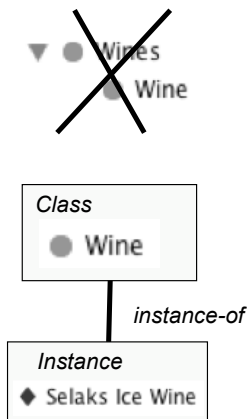
What's in a name? (II)

- *Capitalization and delimiters*
 - *Use spaces: Meal course*
 - *Run words together: MealCourse*
 - *Use underscore or dash: Meal_Course*
- *Singular or plural*
 - *Be consistent*
- *Prefix and suffix conventions*
 - *Common for properties: has-maker, has-winery*
 - *Wine rather than Wine class*
 - *Consistency: if Red **wine**, then White **wine***

Classes and their names

- *Classes represent concepts in the domain, not their names*
- *The class name can change, but it will still refer to the same concept*
- *Synonym names for the same concept are not different classes*
 - *Many systems allow listing synonyms as part of the class definition*

Single and plural class names



- *A “wine” is not a kind-of “wines”*
- *A wine is an instance of the class Wines*
- *Class names should be either*
 - *all singular*
 - *all plural*

Going Deeper

- *Defining classes and a class hierarchy*
- *Deciding when to create a new class*
- *Defining constraints on slots*
- *Defining instances (forms)*
- *Naming conventions*
- *General considerations*

Limiting the scope (II)

- *Ontology of wine, food, and their pairings probably will not include*
 - *Bottle size*
 - *Label color*
 - *My favorite food and wine*
- *An ontology of biological experiments will contain*
 - *Biological organism*
 - *Experimenter*
- *Is the class Experimenter a subclass of Biological organism?*

Limiting the scope

- *An ontology should not contain all the possible information about the domain*
 - *No need to specialize or generalize more than the application requires*
 - *No need to include all possible properties of a class*
 - *Only the most salient properties*
 - *Only the properties that the applications require*

Advanced Topics

- *Metaclasses*
- *N-ary relations (reified relations)*

Metaclasses

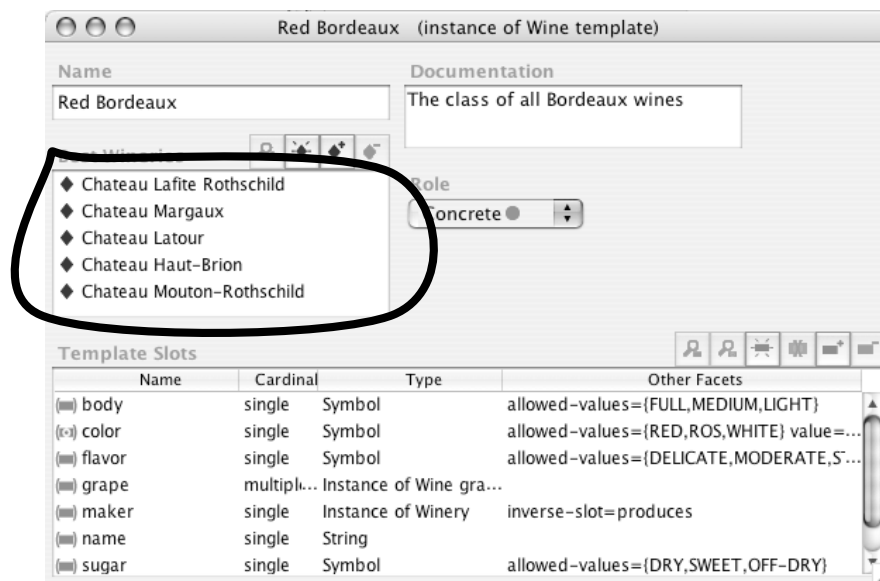
Metaclasses: Templates for class definitions

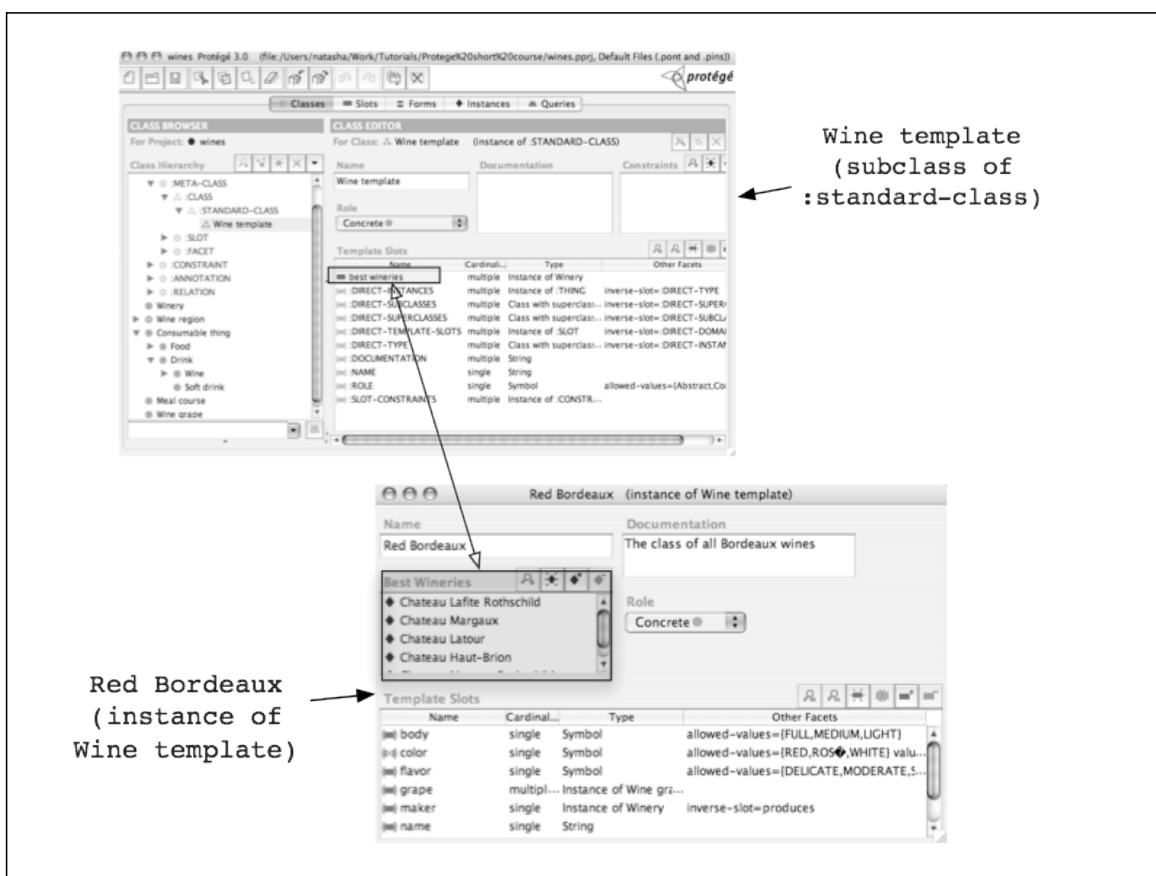
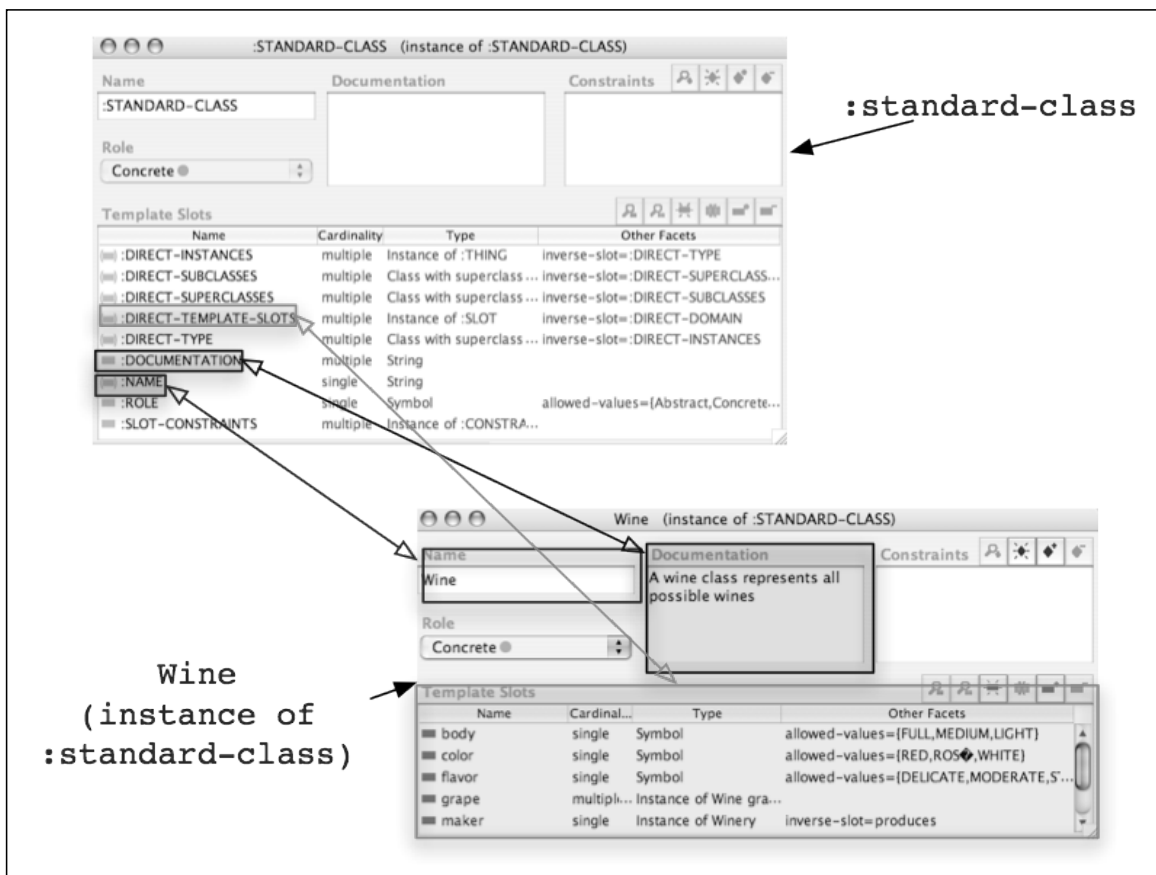
- *Metaclasses enable us to add attributes to class definitions*
- *By default, we have:*
 - *Class name*
 - *Documentation*
 - *Slots*
 - *...*

Metaclasses (II)

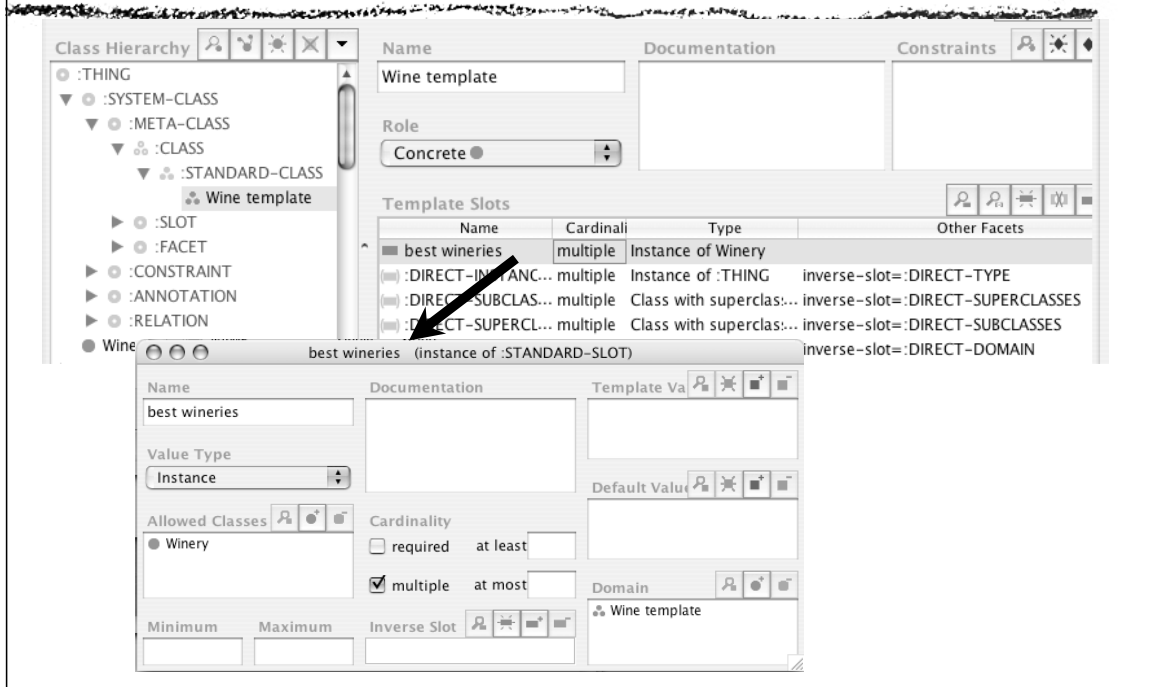
- *Additional attributes:*
 - *Synonyms*
 - *Concept identifier*
 - *Common counter-indications for a drug*
 - *Other class-level properties*

Best Wineries



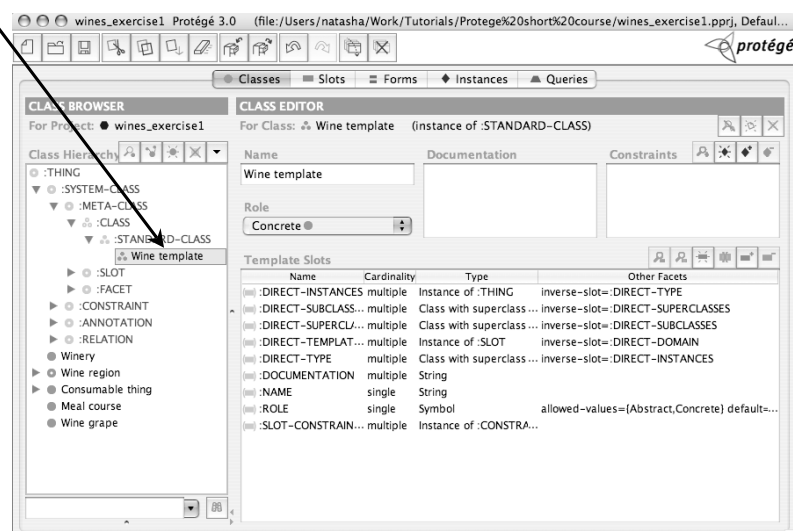


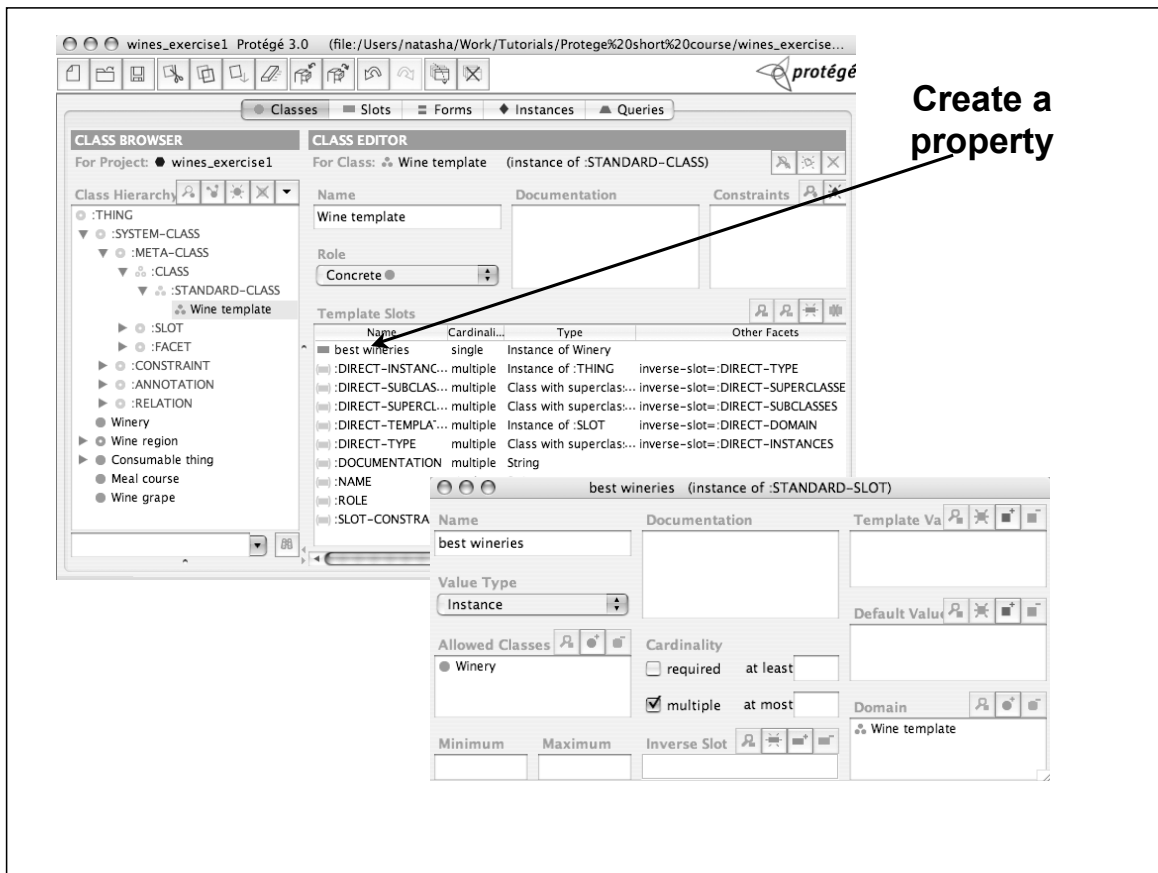
Defining a metaclass



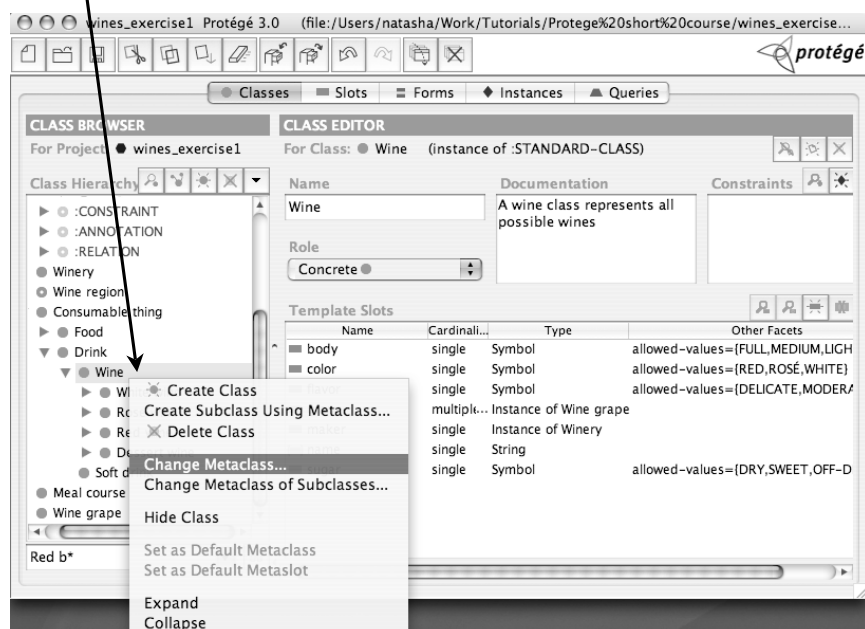
Exercise: create a metaclass

Choose a new metaclass





**Change the metaclass
for Wine
(right-click)**

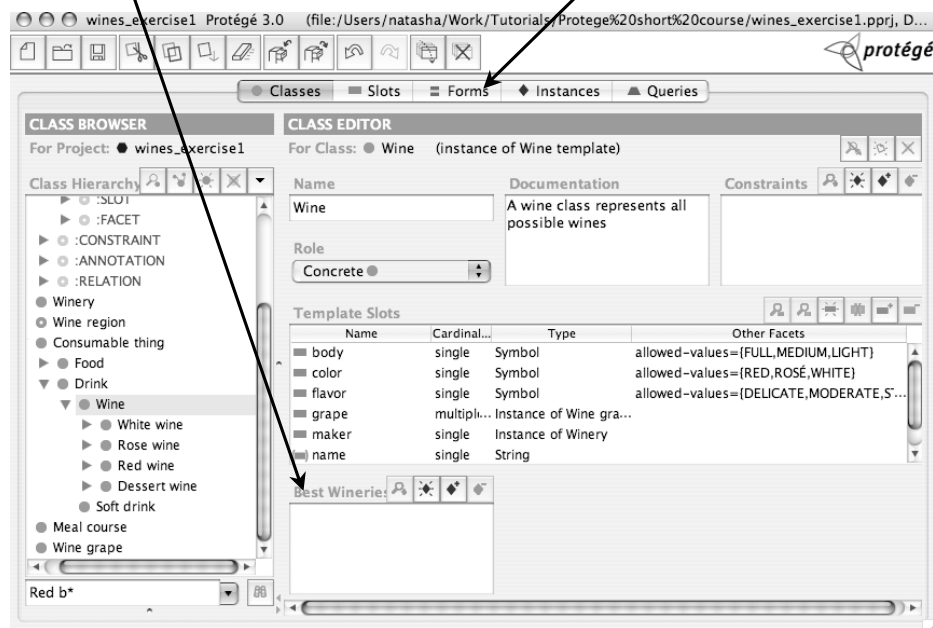


Select a new
metaclass

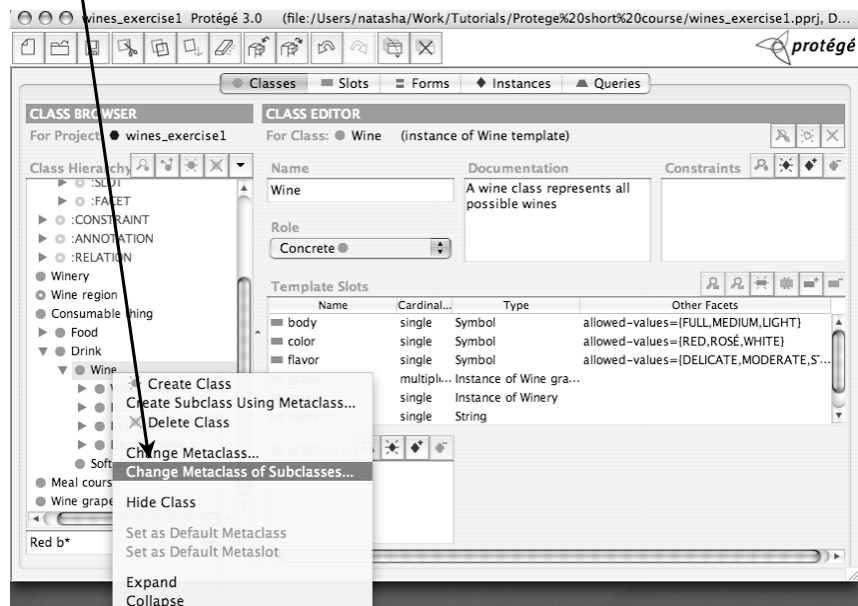


Class Wine now
has a new slot

Use the Forms tab
to change the position
of the slot on the form



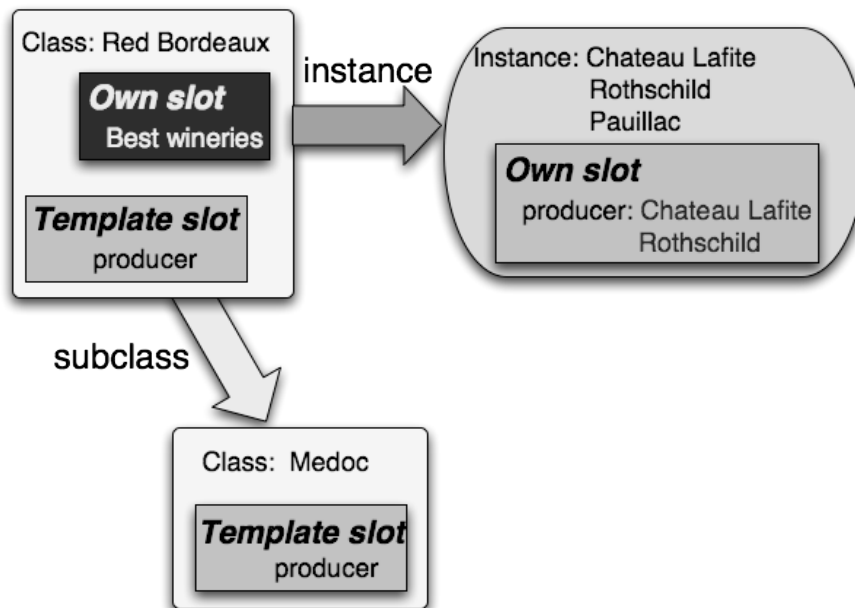
**Change the
metaclass for all
Wines**



Template Slots vs Own Slots

- *Template slots:*
 - *apply to a class*
 - *describe properties of class instance*
- *Own slots:*
 - *apply to an instance (class is also an instance)*
 - *describe properties of the instance (class) itself*

Template and Own Slots: Inheritance



Template and Own Slots: Inheritance

- *Template slots*
 - *inherited to subclasses*
 - *usually given values at instances*
- *Own slots*
 - *not inherited*

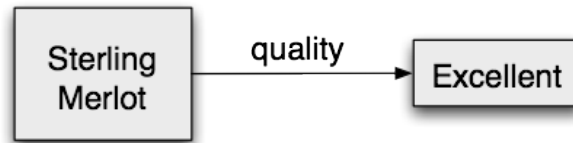
Where do they come from?

- *Template slots*
 - *inherited from a superclass*
 - *defined at class*
- *Own slots*
 - *defined as template slots at the type*

Using N-ary Relations

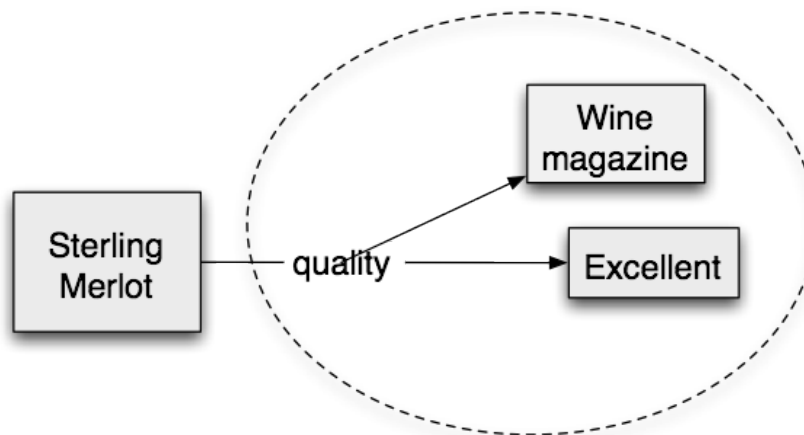
N-ary Relations

Binary Relation

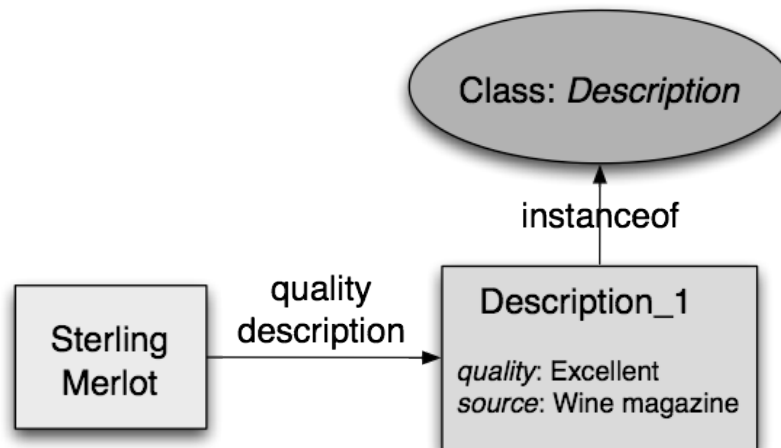


- *According to whom?*

Adding attributes to a Relation



Define a class for a relation: Reification



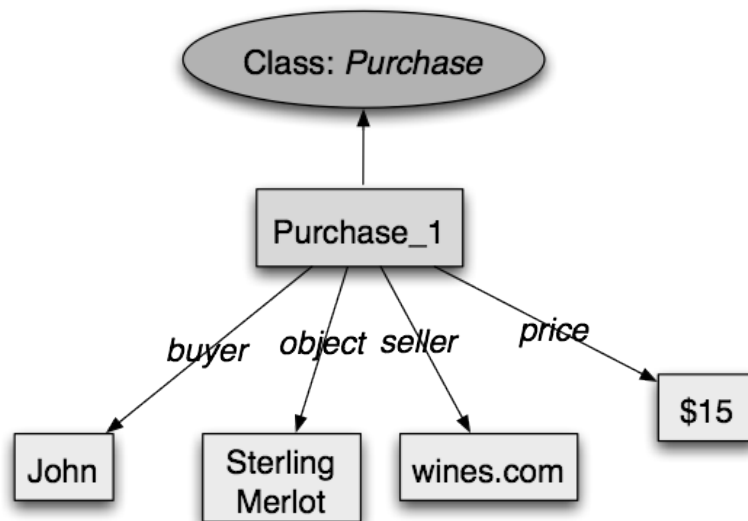
A Relation Between Multiple Participants

John buys Sterling Merlot from wines.com for \$15

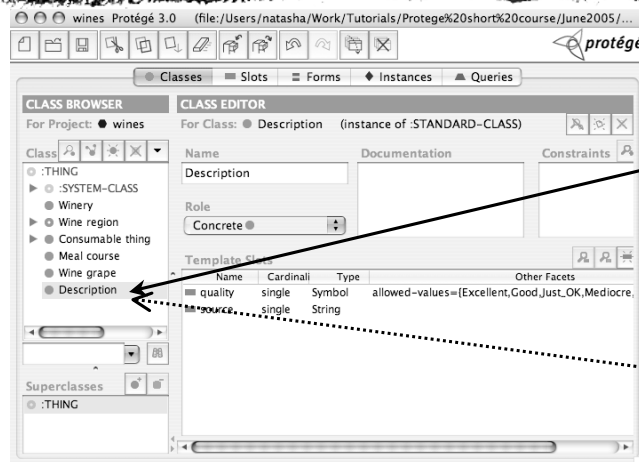
◦ *Participants in this relation:*

- *John*
- *Sterling Merlot*
- *wines.com*
- *\$15*

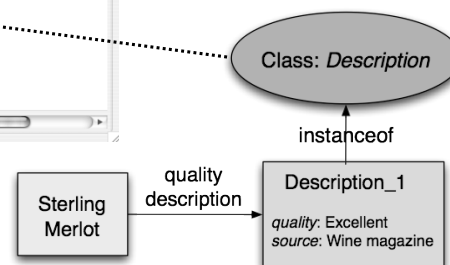
Network of Participants



Exercise: add n-ary relations



Choose a class
for relation



Add description slot to class Wine

Class: *Description*

instanceof

Description_1

quality description

Sterling Merlot

quality: Excellent
source: Wine magazine

Add instances of n-ary relation

Class: *Description*

instanceof

Description_1

quality description

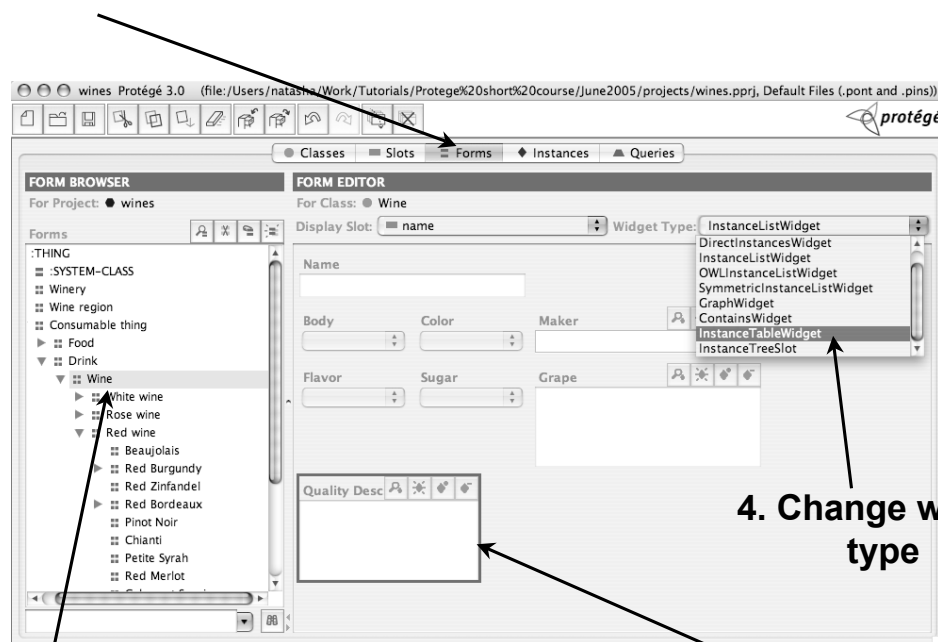
Sterling Merlot

quality: Excellent
source: Wine magazine

Using InstanceTableWidget

- *Use InstanceTableWidget and InstanceRowWidget for n-ary relations*
 - *hides the extra level of indirection*
 - *allows in-place editing*

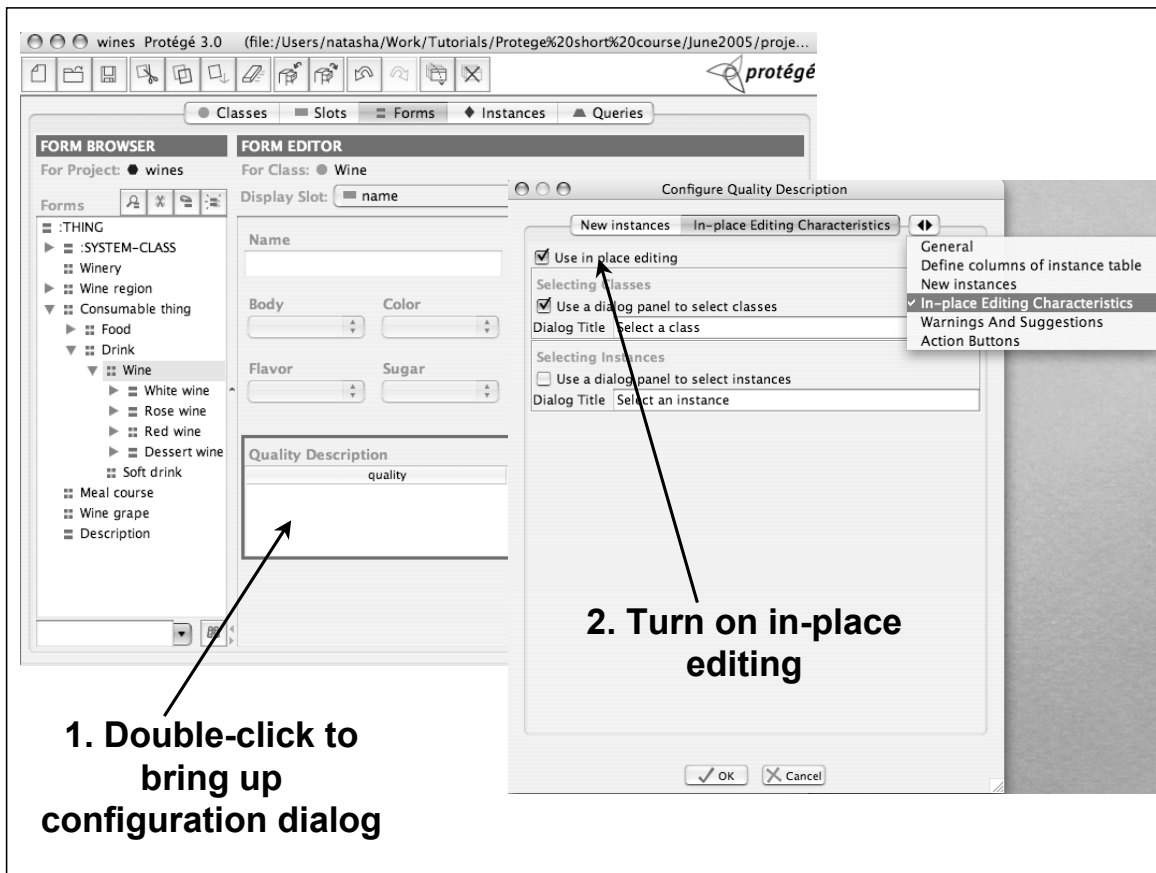
1. Select Forms tab



2. Select class
Wine

3. Select the
widget for Quality
description

4. Change widget
type



Sterling Merlot (instance of Red Merlot, internal name is wines...)

Name: Sterling Merlot

Maker: Sterling Vineyards

Body: FULL Color: RED

Flavor: STRONG Sugar: DRY

Tannin Level: [Slider]

Grape: Merlot grape

Quality Description

quality	source
Excellent	Wine magazine
Just_OK	Wine spectator

